ARM Core
Cortex-A8 (AT400/AT401/AT490)
Errata Notice

This document contains all errata known at the date of issue in supported releases up to and including revision r3p2 of Cortex-A8 Optimized Implementation

Proprietary notice

Document confidentiality status

This document is Non Confidential.

Web address

http://www.arm.com/

Feedback on the product

If you have any comments or suggestions about this product, contact your supplier giving:

- The product name
- A concise explanation of your comments.

Feedback on this document

If you have any comments on about this document, please send email to mailto:support-cores@arm.com giving:

- The document title
- The documents number
- The page number(s) to which your comments refer
- A concise explanation of your comments

General suggestion for additions and improvements are also welcome.

Contents

Introduction

Scope

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

- a unique defect tracking identifier
- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behaviour occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible

Categorisation of Errata

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

Category 1	Behaviour that is impossible to work around and that severely restricts the use of the product in all, or the majority of applications, rendering the device unusable.
Category 2	Behaviour that contravenes the specified behaviour and that might limit or severely impair the intended use of specified features, but does not render the product unusable in all or the majority of applications.
Category 3	Behaviour that was not the originally intended behaviour but should not cause any problems in applications.
Implementation	Errata that are of particular interest to those implementing the product and that have no software implications
System	Errata or possible issues that have system implications and therefore should be considered by system designers
Testchip	Errata that are identified with components in the Testchip RTL that make up the validation testbench such as the MBIST Controller and AXI components.

Change Control

The Cortex-A8 product revisions r1p0 and r2p0 are obsolete. Errata that affect only r1p0 or r2p0 revisions have been removed from this errata document.

09 Apr 2010: Changes in Document v20

Page	Status	ID	Cat	Summary
25	Updated	430973	Cat 2	Stale prediction on replaced interworking branch causes Cortex-A8 to execute in the wrong ARM/Thumb state
37	Updated	454179	Cat 2	Stale prediction may inhibit target address misprediction on next predicted taken branch
53	Updated	628216	Cat 2	If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost
57	New	728018	Cat 2	Cache maintenance operations by MVA for a non-cacheable memory region can result in processor deadlock
80	New	728021	Cat 3	Load and stores executed in debug state can result in memory corruption when force write through is enabled in the DSCCR

04 Nov 2009: Changes in Document v19

Page	Status	ID	Cat	Summary
30	Updated	451034	Cat 2	An instruction sequence including a Neon store to L1 cacheable memory can cause deadlock or unexpected abort
44	Updated	586320	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC are not guaranteed to push data to external memory.
79	Updated	715847	Cat 3	VCVT.f32.u32 can return the wrong result for the input 0xfff_ff01in one specific configuration of floating point unit
84	Updated	488063	System	ARPROT[0] is incorrectly set to indicate a USER transaction for memory accesses generated from user tablewalks
56	New	725233	Cat 2	PLD instructions executed with PLD data forwarding enabled can result in a processor deadlock

06 May 2009: Changes in Document v18

Page	Status	ID	Cat	Summary
18	Updated	657417	Cat 1	A 32-bit branch instruction that spans two 4K regions can result in an incorrect instruction fetch or processor deadlock
25	Updated	430973	Cat 2	Stale prediction on replaced interworking branch causes Cortex-A8 to execute in the wrong ARM/Thumb state
37	Updated	454179	Cat 2	Stale prediction may inhibit target address misprediction on next predicted taken branch
55	New	709718	Cat 2	Load and store operations to shared device memory regions may not complete in program order

79	New	715847	Cat 3	VCVT.f32.u32 can return the wrong result for the input 0xfff_ff01in one specific configuration of floating point unit
76	New	687067	Cat 3	BTB invalidate by MVA operations will not work as intended when the IBE bit is enabled
78	New	693270	Cat 3	Taking a watchpoint is incorrectly prioritized over a precise data abort if both occur simultaneously on the same address

26 Jan 2009: Changes in Document v17

Page	Status	ID	Cat	Summary
18	New	657417	Cat 1	A 32bit branch instruction that spans 2 4K regions can result in an incorrect instruction fetch or processor deadlock

20 Nov 2008: Changes in Document v16

Page	Status	ID	Cat	Summary
51	Updated	621816	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC can result in processor deadlock
53	Updated	628216	Cat 2	If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost.

17 Nov 2008: Changes in Document v15

Page	Status	ID	Cat	Summary
50	New	621766	Cat 2	Under a specific set of conditions, executing a sequence of NEON or vfp load instructions can cause processor deadlock
51	New	621816	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC can result in hanging the processor
53	New	628216	Cat 2	If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost.

02 Oct 2008: Changes in Document v14

Page	Status	ID	Cat	Summary
44	Updated	586320	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC are not guaranteed to push data to external memory.

09 Sep 2008: Changes in Document v13

Page	Status	ID	Cat	Summary
44	New	586320	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC are not guaranteed to push data to external memory.
46	New	586323	Cat 2	Cache clean memory ops generated by either the Preload Engine or Clean by MVA to PoC instructions may corrupt memory
48	New	586324	Cat 2	Under a specific set of conditions, a cache maintenance operation done by MVA can result in a corruption of memory
74	New	588115	Cat 3	A RAW hazard on certain CP15 registers can result in a stale register read

20 Feb 2008: Changes in Document v12

Page	Status	ID	Cat	Summary
73	New	507113	Cat 3	A Neon store to device memory can result in dropping a previous store

25 Jan 2008: Changes in Document v11

Page	Status	ID	Cat	Summary
43	New	485963	Cat 2	CP15 Cache Selection Register (CSSELR) is not banked
72	New	495515	Cat 3	Synchronized entry into debug state from multiple cores in a MP system could result in a debug livelock loop
84	New	488063	system	ARPROT[0] is incorrectly set to indicate a USER transaction for memory accesses generated from user tablewalks

15 Nov 2007: Changes in Document v10

Page	Status	ID	Cat	Summary
40	New	468413	Cat 2	incorrect L2 cache eviction can occur when L2 is configured as an inner cache
41	New	468414	Cat 2	NEON load data can be incorrectly forwarded to a subsequent request
42	New	468416	Cat 2	Under a specific set of conditions, processor deadlock can occur when the L2 cache is servicing write allocate memory
71	New	468415	Cat 3	Swap instruction, a preload instruction, and an instruction fetch request can interact and cause deadlock.

23 Aug 2007: Changes in Document v8

Page	Status	ID	Cat	Summary
37	New	454179	Cat 2	Stale prediction may inhibit target address misprediction on next predicted taken branch

21 Aug 2007: Changes in Document v7

Page	Status	ID	Cat	Summary
33	New	453366	Cat 2	Neon stores to same PA in secure and non-secure may corrupt memory
35	New	453369	Cat 2	Neon store to physical address in lowest 4KB of memory can lead to incorrect results
70	New	453297	Cat 3	BKPT on 2nd half of Thumb-2 instruction cause switch to ARM state.

09 Aug 2007: Changes in Document v6

Page	Status	ID	Cat	Summary
29	New	451027	Cat 2	Neon loads or stores may incorrectly report a precise data abort under certain conditions
28	New	447862	Cat 2	Unaligned store that hits a cache linefillbuffer with errors can cause a later store to corrupt data
30	New	451034	Cat 2	An instruction sequence including a Neon store to L1 cacheable memory can cause deadlock or unexpected abort
31	New	451065	Cat 2	A Neon store instruction to strongly ordered memory can corrupt the data of a previous Neon store
32	New	451325	Cat 2	A double bit ECC error can be masked if a store is done to the data before a load
61	New	434613	Cat 3	Incorrect fault status reported for small pages with the XN attribute set
63	New	436567	Cat 3	CP15 Instruction Set Attribute Register 4, bits 23:20, read 0x1
64	New	436613	Cat 3	ETM OS save sequence resets ETM state
65	New	441933	Cat 3	Incorrect ID register values

66	New	441934	Cat 3	TLB page table walks and Neon load/stores pollute the cache while in debug state
68	New	451020	Cat 3	A Neon store to device memory can result in dropping a previous store
69	New	451329	Cat 3	A Non-secure invalidate operation can generate an abort response when trying to access a secure line
83	New	451335	system	ETM may not power up correctly after a power down sequence

25 Apr 2007: Changes in Document v5

Page	Status	ID	Cat	Summary
25	New	430973	Cat 2	Stale prediction on replaced interworking branch causes Cortex-A8 to execute in the wrong ARM/Thumb state
60	New	430312	Cat 3	A small portion of the ARM opcode space is erroneously decoded as PLI rather than UNDEFINED

23 Mar 2007: Changes in Document v4

Page	Status	ID	Cat	Summary
21	New	424733	Cat 2	Data TLB Preload operations will always be done on the AXI bus as a user privilege level transaction
22	New	424736	Cat 2	NonSecure (NS) version of banked L2_enable is incorrectly used for stores in S-state
23	New	424810	Cat 2	Partial processor reset results in bit 29 of the Debug Status and Control Register (DSCR) resetting incorrectly
59	New	424735	Cat 3	Preload engine transfers can delay draining of write buffer

Errata Summary Table

The errata associated with this product affect product versions as below.

A cell shown thus **X** indicates that the defect affects the revision shown at the top of that column.

ID	Cat	Summary of Erratum	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
416257	Implementation	SAFESHIFTRAM input is not used correctly when doing a partitioned test flow	X	X	X	X								
451335	System	ETM may not power up correctly after a power down sequence	X	X	X	X								
488063	System	ARPROT[0] is incorrectly set to indicate a USER transaction for memory accesses generated from user tablewalks	X	X			X			X				
657417	Cat 1	A 32bit branch instruction that spans two 4K regions can result in an incorrect instruction fetch or processor deadlock	X	X	X		X	X	X			X		
424733	Cat 2	Data TLB Preload operations will always be done on the AXI bus as a user privilege level transaction	X	X	X	X								
424736	Cat 2	NonSecure (NS) version of banked L2_enable is incorrectly used for stores in S-state	X	X	X	X								
424810	Cat 2	Partial processor reset results in bit 29 of the Debug Status and Control Register (DSCR) resetting incorrectly	X	X	X	X								
430973	Cat 2	Stale prediction on replaced interworking branch causes Cortex-A8 to execute in the wrong ARM/Thumb state	X	X	X	X								
447862	Cat 2	Unaligned store that hits a cache linefillbuffer with errors can cause a later store to corrupt data	X	X	X	X								

ID	Cat	Summary of Erratum	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
451027	Cat 2	Neon loads or stores may incorrectly report a precise data abort under certain conditions	X	X	X	X								
451034	Cat 2	An instruction sequence including a Neon store to L1 cacheable memory can cause deadlock or unexpected abort	X											
451065	Cat 2	A Neon store instruction to strongly ordered memory can corrupt the data of a previous Neon store	X	X	X	X								
451325	Cat 2	A double bit ECC error can be masked if a store is done to the data before a load	X	X	X	X								
453366	Cat 2	Neon stores to same PA in secure and non-secure may corrupt memory	X	X	X	X								
453369	Cat 2	Neon store to physical address in lowest 4KB of memory can lead to incorrect results	X	X	X	X								
454179	Cat 2	Stale prediction may inhibit target address misprediction on next predicted taken branch	X	X	X	X								
468413	Cat 2	incorrect L2 cache eviction can occur when L2 is configured as an inner cache					X	X	X	X	X	X	X	X
468414	Cat 2	NEON load data can be incorrectly forwarded to a subsequent request					X	X	X	X	X			
468416	Cat 2	Under a specific set of conditions, processor deadlock can occur when the L2 cache is servicing write allocate memory					X			X				
485963	Cat 2	CP15 Cache Selection Register (CSSELR) is not banked	X	X	X	X	X			X				
586320	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC are not guaranteed to push data to external memory.					X	X		X	X			

ID	Cat	Summary of Erratum	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
586323	Cat 2	Cache clean memory ops generated by either the Preload Engine or Clean by MVA to PoC instructions may corrupt memory					X	X		X	X			
586324	Cat 2	Under a specific set of conditions, a cache maintenance operation done by MVA can result in a corruption of memory	X	X	X	X	X	X		X	X			
621766	Cat 2	Under a specific set of conditions, executing a sequence of NEON or vfp load instructions can cause processor deadlock	X	X	X	X								
621816	Cat 2	Clean and Clean/Invalidate maintenance ops by MVA to PoC can result in processor deadlock							X					
628216	Cat 2	If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost.	X	X	X	X	X	X	X	X	X	X	X	X
709718	Cat 2	Load and store operations to shared device memory regions may not complete in program order					X	X	X	X	X	X	X	
725233	Cat 2	PLD instructions executed with PLD data forwarding enabled can result in a processor deadlock	X	X	X	X								
728018	Cat 2	Cache maintenance operations by MVA for a non-cacheable memory region can result in processor deadlock	X	X	X	X	X	X	X	X	X	X	X	X
424735	Cat 3	Preload engine transfers can delay draining of write buffer	X	X	X	X								
430312	Cat 3	A small portion of the ARM opcode space is erroneously decoded as PLI rather than UNDEFINED	X	X	X	X								
434613	Cat 3	Incorrect fault status reported for small pages with the XN attribute set	X	X	X	X								

ID	Cat	Summary of Erratum	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
436567	Cat 3	CP15 Instruction Set Attribute Register 4, bits 23:20, read 0x1	X	X	X	X								
436613	Cat 3	ETM OS save sequence resets ETM state	X	X	X	X								
441933	Cat 3	Incorrect ID register values	X	X	X	X								
441934	Cat 3	TLB page table walks and Neon load/stores pollute the cache while in debug state	X	X	X	X								
451020	Cat 3	A Neon store to device memory can result in dropping a previous store	X	X	X	X								
451329	Cat 3	A Non-secure invalidate operation can generate an abort response when trying to access a secure line	X	X	X	X								
453297	Cat 3	BKPT on 2nd half of Thumb-2 instruction cause switch to ARM state.	X	X	X	X								
468415	Cat 3	Swap instruction, a preload instruction, and an instruction fetch request can interact and cause deadlock.					X			X				
495515	Cat 3	Synchronized entry into debug state from multiple cores in a MP system could result in a debug livelock loop	X	X	X	X	X	X	X	X	X			
507113	Cat 3	A Neon store to device memory can result in dropping a previous store	X	X	X	X	X	X	X	X	X			
588115	Cat 3	A RAW hazard on certain CP15 registers can result in a stale register read					X	X	X	X	X			
687067	Cat 3	BTB invalidate by MVA operations will not work as intended when the IBE bit is enabled	X	X	X	X	X	X	X	X	X	X	X	X
693270	Cat 3	Taking a watchpoint is incorrectly prioritized over a precise data abort if both occur simultaneously on the same address					X	X	X	X	X	X	X	X

ID	Cat	Summary of Erratum	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
715847	Cat 3	VCVT.f32.u32 can return the wrong result for the input 0xfff_ff01in one specific configuration of floating point unit					X	X	X	X	X	X	X	X
728021	Cat 3	Load and stores executed in debug state can result in memory corruption when force write through is enabled in the DSCCR	X	X	X	X								

Errata - Category 1

657417: A 32bit branch instruction that spans two 4K regions can result in an incorrect instruction fetch or processor deadlock

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 1, Present in: r1p1,r1p2,r1p3,r2p1,r2p2,r2p3,r3p0, Fixed in r1p7,r2p5,r2p6, r3p1.

Description

If, while executing code in Thumb or ThumbEE state, a 32-bit Thumb-2 branch instruction is executed that spans two 4KB regions (i.e. the PC of the branch is xxxxxFFE), and the target address of the branch falls within the first region, it is possible for the processor to incorrectly determine that the branch instruction and the branch target are both within the same 4KB region and therefore a TLB lookup is not required on the target address. This failure to do a TLB lookup and instead use the physical address of the branch instruction for bits [31:12] causes the processor to branch to the wrong instruction address. It is also possible in some cases for the processor to enter a deadlock state when this incorrect fetch address is generated.

NOTE: This erratum corresponds to bug #5102 in the ARM internal Bugzilla database.

Conditions

1. The processor is executing in Thumb or ThumbEE state and the instruction stream includes the Thumb-2 extensions
2. A 32-bit opcode Thumb-2 direct branch (BLX.W, BL.W, B.W, Bcc.W) that spans two 4KB regions exists
3. The target of said branch is in the same 4KB region as the first halfword of the branch
4. The branch is issued in pipe 1 (which means it must dual issue with a prior instruction which will reside in pipe 0)
5. The instruction immediately preceding the branch must be a 32-bit length non-branch instruction. (i.e. this is the instruction issued in pipe0 in parallel with the branch)
6. Program flow prediction is enabled (CR1.Z bit = 1)
7. More than one iteration of the branch occurs so that the branch will be resident in the BTB
8. The branch must be predicted taken
9. The page and line for both halves of the branch must be resident in the TLB and I-cache respectively but not in the BTB, when the branch is first encountered

Additional conditions needed for the deadlock scenario

1. The L1 instruction cache must be a 32k cache and not a 16k cache
2. The branch must be encountered at least 3 times

Implications

If the processor jumps to an incorrect fetch address, that process will run an incorrect code sequence until it triggers an exception condition or the process is terminated by the operating system. If the deadlock condition occurs, it can only be interrupted by pulling the RESET pin on the processor.

This erratum is not possible for 32-bit ARM code or 16-bit Thumb code and is limited to Thumb-2 and ThumbEE code.

Workaround

Three potential software workarounds are possible for this erratum:

1. Force single issue in the processor. Forcing single issue can be done by setting bit[10] in the Auxiliary Control Register. Forcing single issue will significantly degrade processor performance, the impact of which will vary depending on the application.
2. Avoidance of the generation of 32-bit branches that span two 4KB regions. This workaround would require modifications to the s/w tool chain. Note that addresses are not usually finalized until final binary creation in the linking stage and changing them could impact other PC relative address calculations used in the program.
3. Avoid conditions (4) and (5) from ever occurring together. This could be done by having the compiler ensure that the instruction immediately preceding the 32-bit branch is always either another branch instruction or is a 16-bit instruction. This workaround does not require any knowledge of instruction alignment.

Errata - Category 2

424733: Data TLB Preload operations will always be done on the AXI bus as a user privilege level transaction

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7 Fixed in r2p1.

Description

Any cp15 TLB preload operation that requires an external memory transfer to complete the tablewalk will do the AXI transaction at a user privilege level rather than supervisor privilege level (APROT[0] = 0).

NOTE: This erratum corresponds to bug #4596 in the ARM internal Bugzilla database

Conditions

1. The system makes use of the ARPROT and AWPROT bits to filter AXI transactions
2. A data TLB preload operation is done to a privileged region of memory

Implications

If the system uses the ARPROT user/privilege bit to filter AXI transactions, and the region is considered privileged, then a table walk to this region will return data abort. Note that this erratum only effects data TLB preload operations and not instruction TLB preload operations.

Workaround

No software workaround is required unless the system makes use of APROT[0] to filter AXI transactions based on privilege level. In such a system, the software workaround would be to use a normal load instruction instead of the cp15 data TLB preload operation. One load instruction to any address in the page to be preloaded into the TLB will place that address in the TLB. All other code in the TLB lockdown sequence should remain unchanged.

424736: NonSecure (NS) version of banked L2_enable is incorrectly used for stores in S-state

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7 Fixed in r2p1.

Description

If the core is in Secure State, and issues a memory transaction to non-secure memory (page table NS attribute = 1), the banked L2_enable should still come from the Secure version of the corresponding CP15 register. In this case, because of this erratum, the non-secure version of the L2_enable is used for stores to non-secure memory, while the secure version is used for loads.

As a result, if the secure and non-secure versions for L2_enable differ, a coherency problem can occur between secure reads and writes to a region of non-secure cacheable memory. This can lead to loads failing to see result of previous stores to those locations.

The L2 enable was not banked before r1p1, and so this erratum did not occur before r1p1.

NOTE: This erratum corresponds to bug #4608 in the ARM internal bugzilla database.

Conditions:

1. The core is in Secure state
2. The core reading and writing to a region of non-secure memory
3. The Secure L2 Cache enable and the Non-secure L2 cache enable have different values

Implications

If the L2_enable differs between the secure and non-secure versions, even without this erratum, there is a potential coherency problem in the cache when communicating in cacheable memory between the software running on the secure side and software running on the non-secure side, as there is a mismatch in the cache use. For this reason, using non-secure cacheable memory from the secure side when the Secure L2 Cache enable and the Non-secure L2 cache enable have different values should be avoided where possible. If it is not possible, then cache maintenance must be used to resolve the cache coherency issues arising.

This erratum means that when the Secure L2 Cache enable and the Non-secure L2 cache enable have different values, the same cache coherency problem exists if the Secure side is both reading and writing to the same lines of the L2 cache using Non-secure memory.

Workaround

To work around this issue, the secure and non-secure versions of the L2 enable need to be the same value while in secure state if the processor is accessing non-secure cacheable memory in the secure state.

If it is not a requirement for the system to support separate values of the L2 enable for the secure and non-secure worlds, then another workaround would be to leave the L2 enabled for both the secure and non-secure world. For this workaround, you would also want to block non-secure access to the CP15 registers to prevent changing the L2 enable state.

424810: Partial processor reset results in bit 29 of the Debug Status and Control Register (DSCR) resetting incorrectly

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7 Fixed in r2p1.

Description

The architected reset value for DSCR[29] DTRTXfull flag is 0. If the processor is reset by asserting nPORESET without simultaneously asserting PRESETn then DSCR[29] incorrectly reset to 1.

The Cortex-A8 requires that, when it is first powered ON, nPORESET and PRESETn must both be asserted and therefore in this case the erratum does not occur. This is described in Cortex-A8 Technical Reference Manual, figure 10-6.

nPORESET is asserted without the assertion of PRESETn where the core power domain must be reset after power has been restored, but the debug power domain has been left powered up and running. This is most common when the debug functionality is being used on a system which is dynamically powering down and powering up.

If Cortex-A8 OS Save & Restore functionality is being used as described in the Technical Reference Manual and ARMv7 Debug Architecture Reference Manual Supplement then the restore sequence will override the incorrect reset value in DSCR[29], and the erratum will not be seen.

NOTE: This erratum corresponds to bug #4590 in the ARM bugzilla database.

Conditions:

1. nPORESET is being asserted without the assertion of PRESETn
2. The Debug Comms Channel is being used by the debugger

Implications

If the Debugger reads the DSCR register, which has been incorrectly set by the assertion of nPORESET without the assertion of PRESETn, it will erroneously interpret that the DTRTX (Debug Comms Channel, TX direction) is ready. The Debugger may then read this 32-bit register (which will contain an unknown value) and receive corrupted data.

Workaround

If the Cortex-A8 does not have a debug monitor or other code running which uses the debug communications channel, the Debugger can perform a dummy DTRTX register read and discard the value read. This dummy read will automatically clear the DSCR[29] DTRTXfull flag.

If the debugger is attempting to connect with a debug monitor on the Cortex-A8, then this erratum can lead to corruption of this connection. In this case then both the debug monitor and the debugger must implement a protocol that establishes the connection even in the presence of an initial spurious data word. The protocol design must take into account that this spurious data may or may not be present, since the DSCR[29] DTRTXfull resets correctly in some scenarios but incorrectly in others.

A trivial example of that protocol is:

- The debug monitor sends two CONNECT messages (assume each message is a 32-bit word). It expects that one of these may be lost.
- The debugger expects one or two CONNECT packets on connecting to Cortex-A8. The effects of receiving one or two CONNECT packets are identical.

430973: Stale prediction on replaced interworking branch causes Cortex-A8 to execute in the wrong ARM/Thumb state

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7. Fixed in r2p1.

Description

If a code sequence containing an ARM <-> Thumb interworking branch is replaced with another code sequence at the same virtual address, whether due to self-modifying code or virtual to physical address re-mapping, Cortex-A8 does not recover from the stale interworking branch prediction. This results in Cortex-A8 executing the new code sequence in the incorrect ARM or Thumb state.

NOTE: This erratum corresponds to bug #4768 in the ARM bugzilla database.

Conditions:

1. The branch predictor is enabled (CP15 Main Control Register Z bit is 1)
2. The processor executes code containing a predictable interworking branch at a given virtual address. This branch switches between ARM and Thumb state, or between Thumb and ARM state.
3. This code is later replaced by different code not containing a branch at the given virtual address.

Predictable interworking branches in ARM state are:

1. BLX(1)
2. BLX(2)
3. BX
4. LDM(1) with the PC in the register list
5. LDR with the PC as the destination register
6. ALU instructions with PC as the destination register.

Examples of ALU instructions with PC as the destination register are:

1. ADD pc,r0,r1, LSL #2
2. MOV pc, r14

Predictable interworking branches in Thumb state are:

1. BLX(1)
2. BLX(2)
3. BX
4. POP with the PC in the register list
5. LDM with the PC in the register list
6. LDR with the PC as the destination register.

Implications

Self-modifying code, OS context switches, and security state transitions may not work as expected. The new code may execute in the wrong ARM/Thumb state and, therefore, produce incorrect results. Also, Cortex-A8 may unexpectedly take an exception. For example, an Undefined instruction exception.

Workaround

This erratum can only be reproduced if either the virtual to physical address mapping changes, such as on a context switch, or if self-modifying code is used. In many of these cases, the v7 version of the ARM Architecture specifies that a BTB Invalidate instruction should be executed. However this erratum means that there are some additional cases that need to have a BTB Invalidate.

As a result of this erratum, the branch predictor maintenance operations must be used to invalidate entries in the branch predictor after any of the following events:

- enabling or disabling the MMU
- writing new data to instruction locations
- writing new mappings to the translation tables
- changes to the TTBR0, TTBR1, or TTBCR register.

This requirement to invalidate the BTB in the additional cases described above, is consistent with what was required in the v6 version of the ARM Architecture. Therefore, any operating system that is currently compatible with the v6 version of the ARM Architecture should already include all required BTB Invalidate operations. Only operating systems that have relaxed their usage of BTB Invalidate as part of porting to the v7 version of the ARM Architecture will need to add additional BTB Invalidate instructions to work around this erratum.

In addition to using extra BTB Invalidate instructions as described above, the BTB Invalidate operation must also be enabled. By default, the BTB Invalidate instruction is treated as a NOP on Cortex-A8. However, it is possible to enable the BTB Invalidate instruction such that it actually does a full invalidate of the BTB by setting the IBE bit (bit 6) in the CP15 Auxiliary Control Register. As a consequence of erratum 687067, the L1 System Array Debug Register should be cleared to 0 before the IBE bit is set using the following code sequence:

```
MOV r1, #0
MCR p15, 0, r1, c15, c1, 0   ; write instruction data 0 register
MRC p15, 0, R1, c1, c0, 1    ; read Aux Ctl Register
ORR R1, R1 #(1 << 6)         ; set IBE to 1
MCR p15, 0, R1, c1, c0, 1    ; write Aux Ctl Register
```

The above code needs to be executed in Secure state. ARM Limited recommends that this code is added to the boot monitor.

Systems utilizing TrustZone technology require additional BTB invalidation during transitions between the secure and non-secure worlds. A BTB Invalidate All instruction should be placed as soon as possible after the entry point in the secure monitor code and immediately preceding any returns to the nonsecure world. However, due to the nature of secure state entry via multiple exception vectors, it may not be possible to insert a BTB Invalidate opcode until after the initial jump to the vector handler occurs. To avoid exposing the erratum during the initial secure state entry, the secure state branch prediction enable bit (SCTLR.Z bit) should be cleared upon exit of secure state. After execution of the BTB Invalidate All instruction upon entry to the secure world, branch prediction can then be re-enabled. To facilitate the ability to write the secure SCTLR register, CP15SDISABLE must not be asserted.

```
MVBAR
       NOP
       NOP
       LDR    PC, [PC, #48]              ; jump to SMI handler
```



```
        LDR    PC, [PC, #4c]             ; jump to trapped external prefetch abort handler
        LDR    PC, [PC, #50]             ; jump to trapped external data abort handler
        NOP
        LDR    PC, [PC, #58]             ; jump to secure IRQ handler
        LDR    PC, [PC, #5c]             ; jump to secure FIQ handler

smi_handler

        MCR    p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
        MRC    p15,0,r1,c1,c1,0          ; Read CP15 Secure Configuration Register
        BIC    r1, r1, #1                ; clear SCR.NS bit
        MCR    p15,0,r1,c1,c1,0          ; Write CP15 Secure Configuration Register
        MRC    p15, 0, r1, c1, c0, 0     ; read secure SCTLR
        ORR    r1, r1, #0x800            ; turn branch prediction (Z) bit on
        MCR    p15, 0, r1, c1, c0, 0     ; write secure SCTLR

        <handler body>

        MRC    p15, 0, r1, c1, c0, 0     ; read secure SCTLR
        BIC    r1, r1, #0x800            ; turn branch prediction (Z) bit off
        MCR    p15, 0, r1, c1, c0, 0     ; write secure SCTLR
        MCR    p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
        MOVS   pc, r14
```

Systems requiring the CP15SDISABLE bit to be set cannot use this facility, as access to the secure SCTLR is denied. Two alternative options exist:

1. Embargo the non-secure memory region with VA equal to the secure monitor code. This requires participation of the operating system on the non-secure side.
2. Modify the secure monitor code to enter in Thumb state by setting the secure SCTLR.TE bit, and ensure that only 16-bit Thumb branches are encountered until the BTB Invalidate All can be executed. This is done by inserting a small Thumb code region immediately following the vector table (due to the limited range of Thumb

```
MVBAR
        NOP
        NOP
        NOP
        NOP
        B      short_jmp_smi
        NOP
        B      short_jmp_pabt
        NOP
        ...

short_jmp_smi
        MCR.W  p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
        LDR    r0, smi_handler           ; switch to ARM state and jump to handler
        BX     r0
```

This must be done at both the Monitor Base and Secure Vector Base addresses.

447862: Unaligned store that hits a cache linefillbuffer with errors can cause a later store to corrupt data

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If an unaligned store crossing a 64 bit boundary is to the same address as an outstanding cache linefill request that receives an L2 parity, L2 double-bit ECC, or AXI external error response, the store buffer control state will be corrupted when the abort is taken.

If two back to back stores occur, the first one hitting the L1 cache, before this corrupted state in the store buffer is resolved, the first store may go to an incorrect location and corrupt memory.

NOTE: This erratum corresponds to bug #4959 in the ARM internal bugzilla database.

Conditions

1. A cache linefill request is serviced and gets errors on some but not all of the data packet. (This can only occur if L2 parity, L2 ECC, or AXI slaves errors that are due to parity/ECC issues.)
2. An unaligned store crossing a 64 bit boundary hits the fill buffer containing the above cache line with errors

Implications

For systems implementing L2 parity, L2 ECC, or AXI slave devices that can return errors on partial cache lines, memory state may be corrupted if the conditions for this erratum are met.

Workaround

When the erratum occurs, memory corruption can be avoided by clearing out the linefill before performing any subsequent store operations. This can be done by doing a Data Memory Barrier (DMB) instruction at the beginning of any exception handler that can be taken on a data abort.

451027: Neon loads or stores may incorrectly report a precise data abort under certain conditions

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

For a specific sequence of load and store operations, it is possible for an erroneous abort response to be reported on a Neon load or store instruction.

NOTE: This erratum corresponds to bug #4993 in the ARM internal bugzilla database.

Conditions

1. A cache linefill request generated from a L1 data cache miss gets a abort response that is either externally generated from an AXI slave or is a Parity error response from the L2 cache.
2. A subsequent store instruction is executed that fails it's condition codes, but has an address that generates a hash hit, physical address miss to the linefill address that got the abort.
3. The store that fails its condition code check is immediately followed by a Neon load or store that also has a hash hit, physical address miss to the aborting linefill address

When this sequence of events occurs, the Neon load or store instruction at the end of the sequence may incorrectly take a precise abort exception due to the pending abort response associated with the linefill request.

Implications

Getting the erroneous abort on the Neon load or store could result in an incorrect action taken by the abort handler in its attempt to resolve the abort condition.

Workaround

If the system only generates external AXI errors on the first beat of a cache line (decode errors, etc.), the erroneous abort on a Neon load or store can only happen after the expected precise data abort has occurred. Therefore, a workaround is possible which is to add a DMB memory barrier instruction at the beginning of the data abort handler, (and also the monitor handler if trapping to monitor mode on a data abort is a feature that is used by the system).

If the system generates ECC/parity errors from the L2 cache, or partial cache line aborts in the middle of a cache line fill on AXI, then an abort may be reported on the wrong instruction first. For this scenario, there is no workaround. However, this scenario requires an imprecise abort to occur and the general response to an imprecise abort of to save off as much state as possible and shut down. Therefore, the processor is already in an unrecoverable situation.

451034: An instruction sequence including a Neon store to L1 cacheable memory can cause deadlock or unexpected abort

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1, Fixed in r1p2.

Description

If a particular instruction sequence occurs that includes a Neon store operation that hit in the L1 data cache, it is possible that the processor will deadlock or produce an unexpected external abort.

NOTE: This erratum corresponds to bug #5001 in the ARM internal bugzilla database.

Conditions

1. A cacheable Neon store is executed that hits in the L1 data cache
2. The Neon store matches the physical address of the cache line currently in the L1 eviction buffer
3. The Neon store also has the same hash and index as the cache line currently in the fill buffer, but the fill buffer has a different physical address (hash hit/physical address miss)
4. There is an integer store currently in the store buffer that also matches by physical address to the cache line in the fill buffer.

All four of the above memory system hazards must be present at the same time for the deadlock or abort condition to occur.

Implications

If the deadlock state occurs, it can only be exited by asserting the RESET pin on the processor. If an unexpected external abort occurs, processor state is unknown and may not be recoverable.

Workaround

Two possible workarounds exist for this issue. Unfortunately, both workarounds have a visible software impact and may be difficult to implement.

The first workaround is to ensure that no Neon store can hit a line in the L1 eviction buffer by mapping all memory pages used by neon memory operations as inner-non-cacheable. To minimize the performance impact of this work around, the L2 cache should be configured as an outer cache and the Neon L1 caching bit should be disabled. Configuring the L2 cache as an outer cache is done by clearing bit[16] in the L2 Cache Auxiliary Control register. Disabling Neon L1 caching is done by clearing bit[5] of the Auxiliary Control Register. For more information on the Auxiliary Control Register and L2 Cache Auxiliary Control Register, please refer to the Cortex A8 Technical Reference Manual.

The other possible workaround is to ensure that that the Integer Store Buffer is always empty when Neon stores are executed. This can be done by draining the memory system with a Data Memory Barrier (DMB) instruction on every switch between integer memory operations to Neon memory operations. This would only be feasible if Neon code and integer code were segregated and if DMB instructions are placed at the beginning of every exception handler.

451065: A Neon store instruction to strongly ordered memory can corrupt the data of a previous Neon store

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

Given the following sequence of two Neon store instructions:

```
VST <addrA>  (not strongly ordered)
VST <addrB>  (strongly ordered)
```

Where addrB is part of a strongly ordered memory region and addrA is not, it is possible that the Neon store to addrA will incorrectly be marked and treated by the memory system as a strongly ordered store and the Neon store request to addrB will incorrectly be treated as a normal store operation.

NOTE: This erratum corresponds to bug #5004 in the ARM internal bugzilla database.

Conditions

1. execute a Neon store instruction to a memory region that is not strongly ordered, but results in an AXI memory transaction
2. execute a subsequent Neon store instruction to a strongly ordered memory region before the previous store's AXI transaction has been sent out on the bus

Implications

If this erratum occurs, a strongly ordered Neon store can be incorrectly marked as a non-cacheable normal memory request and a normal memory request can be incorrectly marked as strongly ordered. This mistreatment of these requests can result in data corruption. In addition, non-secure Neon stores could potentially interact with Secure Neon stores in this way, allowing non-secure code to corrupt secure memory.

Workaround

The general workaround for this erratum is to not do Neon stores to strongly ordered memory regions. In addition, if there is concern that a security attack could be made using strongly ordered memory regions, this can be avoided by either avoiding the use of Neon stores while in secure mode or by doing a Data Memory Barrier (DMB) instruction upon entry and exit of secure mode.

451325: A double bit ECC error can be masked if a store is done to the data before a load

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If an ECC double bit error occurs on a line of the L2 cache and a store is done to double word that contains the error before the data is read, it is possible that the error will not be reported.

NOTE: This erratum corresponds to bug #4911 in the ARM internal Bugzilla database.

Conditions

1. ECC enabled on L2
2. A line in L2 cache has a double-bit ECC error within a doubleword(ECC is calculated based on a doubleword)
3. A store operation of size less than a doubleword occurs to the doubleword with the double-bit ECC error
4. ECC information is updated based on the store and the double-bit ECC error is lost
5. Subsequent loads of the data and the eviction of the line to memory will not report the double bit ECC error

Implications

Because of this erratum, it is possible that a double bit ECC error in the L2 cache can go undetected. Any load of data containing the incorrect bits will result in incorrect data being used.

Workaround

There is no known workaround for this errata.

453366: Neon stores to same PA in secure and non-secure may corrupt memory

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If secure mode Neon stores are used to access the same physical address in both secure and non-secure memory space, the hazard checking logic in the neon store buffer may not work correctly. Subsequent memory operations may not correctly hazard check against outstanding Neon stores. This could cause the processor to deadlock, load instructions to bring in stale data, or store instructions to execute out of order and corrupt memory.

NOTE: This erratum corresponds to bug #5022 in the ARM internal bugzilla database.

Conditions

1. Processor is in Secure state (NS-Req for this and all subsequent stores = 0)
2. A Neon store NST1 to modify *non-secure* data (NS-Attr = 1) at physical address A1 is executed
3. A Neon store NST2 to modify *secure* data (NS-Attr = 0) with the same physical address A1 is executed
4. A Neon store NST3 to modify *non-secure* data (NS-Attr = 1) at physical address A1 is executed
5. A subsequent memory operation MEM4 (Neon or integer load or integer store) accesses physical address A1 (NS-Attr = 0 or 1)
6. NST1 is resident in the Neon Store Buffer when NST2, NST3, and MEM4 occur with a one cycle gap between NST2 and NST3 (i.e. NST2 occurs in cycle N and NST3 occurs in cycle N+2)

The erratum will also occur if the secure and non-secure memory transactions above are reversed: NST1 and NST3 secure and NST2 non-secure.

Implications

The processor may deadlock, return incorrect data on a load, or corrupt memory on a store.

Possible failure modes:

1. If MEM4 is a load or integer store to address A1 with the same NS-Attr as NST1, the processor will most likely deadlock.
2. If MEM4 is a load or integer store to address A1 with the same NS-Attr as NST2, MEM4 will be executed ahead of NST1 and NST3. Incorrect data may be returned to a register or memory may be corrupted.

Workaround

Ensure that Neon stores to the secure and non-secure versions of the same address aren't executed within a few cycles of each other.

There are several ways to do this:

1. Do not use Neon stores in secure mode to access non-secure data.
2. Do not map the same physical address to different virtual addresses with different values of NS-Attr.
3. If secure Neon stores are used in close proximity to access secure and non-secure copies of the same physical address, ensure that those accesses are separated by a memory barrier (DSB or DMB)

453369: Neon store to physical address in lowest 4KB of memory can lead to incorrect results

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If a Neon store occurs to an address that maps to the lowest 4KB of physical address space, the processor memory hazard checking logic may not work correctly and other memory operations may pass that Neon store without correctly detecting a hazard. As a result of not detecting the memory hazard, incorrect load data could be returned or the memory locations accessed by the Neon store could be corrupted.

NOTE: This erratum corresponds to bug #5031 in the ARM internal bugzilla database.

Conditions

1. A Neon store (NST1) is executed to a physical address in the lowest 4KB of memory (physical address bits[31:12] are all zero).
2. A Neon store (NST2) to the same cache line index, with NST2 physical address bits[11:6] the same as NST1 physical address bits[11:6], is executed.
3. NST2 misses the TLB.
4. A Neon load, integer load, or integer store (MEM3) occurs to the same cache line as NST1.
5. NST2 and MEM3 must both happen while NST1 is still in the Neon store buffer and has not yet been issued to the L2 cache or external memory.

The operation MEM3 should detect a hazard with NST1 in the Neon store buffer and wait for NST1 to complete before issuing. Because of this erratum, the hazard will be missed, and MEM3 may complete to memory before NST1 completes.

Implications

If this erratum occurs, memory could be corrupted, if MEM3 is a store and NST1 overwrites it, or incorrect data could be returned into a processor register, if MEM3 is a load which gets stale data when it should have gotten the result of NST1).

Workaround

The processor must not perform Neon stores to the lowest 4KB of physical memory (physical address bits[31:12] all zero).

How this can be achieved will depend on the OS and software running on the system. Typical possibilities would include:

1. Keep the lowest 4KB of physical address space as an OS reserved page and ensure that the OS does not access it using Neon store instructions.
2. Manage applications using Neon and do not map that page for those applications.

454179: Stale prediction may inhibit target address misprediction on next predicted taken branch

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If a Thumb-2 code sequence containing certain predicted branches is replaced with another ARM or Thumb-2 code sequence at the same virtual address, whether due to self-modifying code or virtual to physical address re-mapping, in certain rare conditions, a subsequent predicted taken branch to an incorrect address will not be flushed. The target address of the replaced branch must also cause a prediction with a target address equal to the same address as the real target of the subsequent predicted branch. The subsequent predicted branch prediction must have an incorrect target address, but must otherwise be properly predicted.

NOTE: This erratum corresponds to bug #5036 in the ARM bugzilla database.

Conditions

1. The branch predictor is enabled (CP15 Main Control Register Z bit is 1).
2. The processor executes code containing a 16-bit length predictable branch at a given virtual address.
3. This code is later replaced by different code containing a 32-bit instruction at the given virtual address.
4. One of the first four instructions at the target of the replaced branch is also a predictable branch to an address A. This code is not executed.
5. The next predicted taken branch in the real code path has a predicted target of address B. The real programmed target address is A.

Implications

Self-modifying code, OS context switches, and security state transitions may not work as expected. The processor will execute code at an incorrect address - in the terminology above, the code at address B will be executed, rather than address A.

Workaround

This erratum can only be reproduced if either the virtual to physical address mapping changes, such as on a context switch, or if self-modifying code is used. In many of these cases, the v7 version of the ARM Architecture specifies that a BTB Invalidate instruction should be executed. However this erratum means that there are some additional cases that need to have a BTB Invalidate. As a result of this erratum, the branch predictor maintenance operations must be used to invalidate entries in the branch predictor after any of the following events:

- enabling or disabling the MMU
- writing new data to instruction locations
- writing new mappings to the translation tables

- changes to the TTBR0, TTBR1, or TTBCR register

This requirement to invalidate the BTB in the additional cases described above, is consistent with what was required in the v6 version of the ARM Architecture. Therefore, any operating system that is currently compatible with the v6 version of the ARM Architecture should already include all required BTB Invalidate operations. Only operating systems that have relaxed their usage of BTB Invalidate as part of porting to the v7 version of the ARM Architecture will need to add additional BTB Invalidate instructions to work around this erratum.

In addition to using extra BTB Invalidate instructions as described above, the BTB Invalidate operation must also be enabled. By default, the BTB Invalidate instruction is treated as a NOP on Cortex-A8. However, it is possible to enable the BTB Invalidate instruction such that it actually does a full invalidate of the BTB by setting the IBE bit (bit 6) in the CP15 Auxiliary Control Register. In addition, branch size mispredicts, which occur when an incorrectly aligned stale prediction is detected, must be completely disabled. This is done by setting the "Disable branch size mispredicts" bit (bit 7) in the CP15 Auxiliary Control Register. This must be done in addition to setting the IBE bit as it is possible to speculatively encounter misaligned branches within a given process if the process contains both ARM and Thumb-2 state code. As a consequence of erratum 687067, the L1 System Array Debug Register should be cleared to 0 before the IBE bit is set.

All of these system register updates can be done with the following code sequence:

```
MOV r1, #0
MCR p15, 0, r1, c15, c1, 0   ; write instruction data 0 register
MRC p15, 0, R1, c1, c0, 1    ; read Aux Ctl Register
ORR R1, R1 #(1 << 6)         ; set IBE and disable branch size mispredict to 1
MCR p15, 0, R1, c1, c0, 1    ; write Aux Ctl Register
```

The above code needs to be executed in Secure state. ARM Limited recommends that this code is added to the boot monitor.

As a consequence of disabling branch size mispredicts, code must not be used that can be interpreted as a 32-bit instruction in ARM state and 16-bit instruction in Thumb state without flushing the BTB. The majority of cases when this scenario is encountered are context switches or self-modifying code, which are enumerated as required flush cases in the list above. An additional case is instructions that can be interpreted in both ARM and Thumb state. The BTB should be flushed each time this code is encountered.

Systems utilizing TrustZone technology require additional BTB invalidation during transitions between the secure and non-secure worlds. A BTB Invalidate All instruction should be placed as soon as possible after the entry point in the secure monitor code and immediately preceding any returns to the nonsecure world. However, due to the nature of secure state entry via multiple exception vectors, it may not be possible to insert a BTB Invalidate opcode until after the initial jump to the vector handler occurs. To avoid exposing the erratum during the initial secure state entry, the secure state branch prediction enable bit (SCTLR.Z bit) should be cleared upon exit of secure state. After execution of the BTB Invalidate All instruction upon entry to the secure world, branch prediction can then be re-enabled. To facilitate the ability to write the secure SCTLR register, CP15SDISABLE must not be asserted.

MVBAR

```
       NOP
       NOP
       LDR    PC, [PC, #48]             ; jump to SMI handler
       LDR    PC, [PC, #4c]             ; jump to trapped external prefetch abort handler
       LDR    PC, [PC, #50]             ; jump to trapped external data abort handler
       NOP
       LDR    PC, [PC, #58]             ; jump to secure IRQ handler
       LDR    PC, [PC, #5c]             ; jump to secure FIQ handler
```

smi_handler

```
       MCR    p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
```



```
MRC    p15,0,r1,c1,c1,0          ; Read CP15 Secure Configuration Register
BIC    r1, r1, #1                ; clear SCR.NS bit
MCR    p15,0,r1,c1,c1,0          ; Write CP15 Secure Configuration Register
MRC    p15, 0, r1, c1, c0, 0     ; read secure SCTLR
ORR    r1, r1, #0x800            ; turn branch prediction (Z) bit on
MCR    p15, 0, r1, c1, c0, 0     ; write secure SCTLR

<handler body>

MRC    p15, 0, r1, c1, c0, 0     ; read secure SCTLR
BIC    r1, r1, #0x800            ; turn branch prediction (Z) bit off
MCR    p15, 0, r1, c1, c0, 0     ; write secure SCTLR
MCR    p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
MOVS   pc, r14
```

Systems requiring the CP15SDISABLE bit to be set cannot use this facility, as access to the secure SCTLR is denied. Two alternative options exist:

1. Embargo the non-secure memory region with VA equal to the secure monitor code. This requires participation of the operating system on the non-secure side.
2. Modify the secure monitor code to enter in Thumb state by setting the secure SCTLR.TE bit, and ensure that only 16-bit Thumb branches are encountered until the BTB Invalidate All can be executed. This is done by inserting a small Thumb code region immediately following the vector table (due to the limited range of Thumb

MVBAR

```
NOP
NOP
NOP
NOP
B      short_jmp_smi
NOP
B      short_jmp_pabt
NOP
...
```

short_jmp_smi

```
MCR.W  p15, 0, r1, c7, c5, 6     ; BPIALL - Invalidate BTB All
LDR    r0, smi_handler           ; switch to ARM state and jump to handler
BX     r0
```

This must be done at both the Monitor Base and Secure Vector Base addresses.

468413: incorrect L2 cache eviction can occur when L2 is configured as an inner cache

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1,r3p2, Open.

Description

It is possible under a specific set of conditions for stale data to be saved into the L2 cache and returned to the processor erroneously on a subsequent load instruction.

NOTE: This erratum corresponds to bug #5042 in the ARM Bugzilla database.

Conditions

1. The L2 cache must be configured as an inner cache rather than as an outer cache
2. The L2 cache must be configured to use write allocate memory type

Implications

If this erratum occurs, stale data could be read by a subsequent load instruction, resulting in incorrect program behaviour.

Workaround

There are two viable workarounds for this erratum. One workaround is to not configure the L2 cache as an inner cache but maintain the default setting as an outer cache. The second workaround is to use the remap registers to remap the inner cache attributes from write allocate to write back instead.

468414: NEON load data can be incorrectly forwarded to a subsequent request

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p2,r2p3,r2p5,r2p6, Fixed in r3p0.

Description

It is possible under a very specific set of conditions for data from a Neon load request to be incorrectly forwarded to a subsequent, unrelated memory request.

NOTE: This erratum corresponds to bug #5044 in the ARM Bugzilla database.

Conditions

1. Neon loads and stores must be in use
2. Neon L1 caching must be disabled
3. Trustzone must be configured and in use
4. The secure memory address space and the non-secure memory address space both use the same physical addresses, either as an alias or the same memory location, or for separate memory locations

Implications

If this erratum is encountered, it is possible for a load request to receive the wrong data value which will likley result in incorrect operation of the program.

Workaround

There are many software solutions for this erratum and only one has to be applied. The recommended solution, if possible, is to map cacheable areas of memory so that both secure and non-secure do not share the same physical address space.

Another possible solution is to force NEON to cache in the L1 data cache. This can be programmed using the Auxiliary Control Register bit[5], L1NEON, as follows:

```
MRC p15, 0, r0, c1, c0, 1  ; read register
ORR r0, r0, #(1<<5)        ; L1NEON caching enable
MCR p15, 0, r0, c1, c0, 1  ; write register
```

Another possible solution is to disable L2 data forwarding from the victim buffers. This can be programmed using the L2 Auxiliary Control Register bit[27], Load data forwarding disable:

```
MRC p15, 1, r0, c9, c0, 2   ; read register
ORR r0, r0, #(1<<27)        ; L2 load data forwarding disable
MCR p15, 1, r0, c9, c0, 2   ; write register
```

Both of these workaround can be implemented with little to no perceived performance impact in the large majority of applications.

468416: Under a specific set of conditions, processor deadlock can occur when the L2 cache is servicing write allocate memory

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p5, Fixed in r2p2,r2p3,r2p6.

Description

If a load request is processed which misses the L2 cache, but cannot be immediately forwarded to the BIU, it will encounter a special hazard which prevents that request from being required to access the L2 cache RAM again to save power. There can be multiple requestors with unique addresses, (i.e. one address per cache line) with this special hazard.

All write-allocate requests that access the L2 cache RAM, on port1, do not have address comparators to check for this special hazard condition. So, if a subsequent write-allocate request is issued to the L2 cache RAM on port1 and allocates a victim buffer, then all requests pending with this special hazard must be forced to perform a L2 cache RAM lookup again to maintain memory coherency.

There is a 1-cycle window in which the write-allocate request must allocate to a victim buffer and a pending request to the BIU is not prohibited from going to the BIU, such that a deadlock can occur.

NOTE: This erratum corresponds to bug #5046 in the ARM Bugzilla database.

Conditions

1. The processor must have L2 cache present and enabled
2. The L2 cache must be configured to support the write allocate memory type

Implications

If this erratum is encounter and processor deadlock occurs, it can only be interrupted by pulling the RESET pin on the processor.

Workaround

The workaround for this erratum is to disable write-allocate by programming the L2 Auxiliary Control Register bit[22], Write allocate disable:

```
MRC p15, 1, r0, c9, c0, 2   ; read register
ORR r0, r0, #(1<<22)        ; Write allocate disable
MCR p15, 1, r0, c9, c0, 2   ; write register
```

Disabling write allocate in the Level 2 cache could have a performance impact for some applications.

485963: CP15 Cache Selection Register (CSSELR) is not banked

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7,r2p1,r2p5, Fixed in r2p2,r2p3,r2p6.

Description

ARMv7 architecture specifies that the CSSELR should be banked between Secure and Non-secure states. Cortex-A8 does not currently bank this register.

NOTE: This erratum corresponds to bug #5050 in the ARM Bugzilla database.

Conditions

1. The system must have an active process in secure state and an active process in non-secure state at the same time.
2. The system must do cache maintenance operations in both secure and non-secure processes.

Implications

A cache cleaning sequence that reads the CSSELR may not work as expected. The published sequence for cleaning the entire cache (ARM Architecture Reference Manual -- ARM DDI 0406A -- section B2.2.4, page B2-17) includes a setting of the CSSELR followed by a reading of the selected Cache Size ID Register (CCSIDR). If the Non-secure side was executing this sequence and there was a secure interrupt between the setting of the CSSELR and the reading of the selected CCSIDR, then there is a possibility the secure side will also use the CSSELR. On returning to the Non-secure side the CSSELR value may have changed, which would make the cache cleaning sequence malfunction. Similarly, a Non-secure interrupt could cause a Secure cache cleaning sequence to malfunction.

Workaround

When transitioning security state, the secure monitor software should save the current CSSELR value (corresponding to the security state the processor is transitioning out of) and restore the previously saved CSSELR value (corresponding to the security state the processor is transitioning into).

586320: Clean and Clean/Invalidate maintenance ops by MVA to PoC are not guaranteed to push data to external memory.

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p2,r2p5,r2p6, Fixed in r2p3,r3p0,

Description

When a Clean to Point of Coherency or Clean and Invalidate to Point of Coherency by MVA operation is done, it is possible that the line will remain present in the L2 cache and any dirty data will not be pushed out on the AXI bus to main memory. This can happen whenever the requested address is present in the L1 cache but not the L2 cache.

NOTE: This erratum corresponds with bug #5066 in the ARM internal bugzilla database.

Conditions

1. The memory region being cleaned is configured in write allocate mode
2. The cache line being cleaned is initially present in the L1 cache and not the L2 cache

Implications

If a clean or clean and invalidate operation does not work as intended and leaves data present in the L2 cache, then memory coherency in the system can no longer be guaranteed. Therefore, this erratum will impact any code sequence that is used to maintain system coherence.

Workaround

There are several possible software workarounds for this erratum. This simplest one to implement is to simply disable write allocate in the Level 2 cache. This can be done with the following instruction sequence:

```
MRC p15, 1, <Rd>, c9, c0, 2      ; read L2 cache Aux Ctrl Reg
ORR <Rd>, <Rd>, #(1 << 22)       ; set the Write Allocate disable bit
MCR p15, 1, <Rd>, c9, c0, 2      ; write the L2 cache Aux Ctrl Reg
```

Disabling write allocate in the Level 2 cache could have a performance impact for some applications. If this performance impact is deemed to be too high, there are two other software workarounds that may be used. The first is to disable write allocate around each sequence of clean by MVA to PoC or clean/invalidate by MVA to PoC instructions:

```
MRC p15, 1, <Rd>, c9, c0, 2      ; read L2 cache Aux Ctrl Reg
ORR <Rd>, <Rd>, #(1 << 22)       ; set the Write Allocate disable bit
MCR p15, 1, <Rd>, c9, c0, 2      ; write the L2 cache Aux Ctrl Reg
<perform sequence of MVA operations here>
MRC p15, 1, <Rd>, c9, c0, 2      ; read L2 cache Aux Ctrl Reg
BIC <Rd>, <Rd>, #(1 << 22)       ; clear the Write Allocate disable bit
MCR p15, 1, <Rd>, c9, c0, 2      ; write the L2 cache Aux Ctrl Reg
```

The final workaround that can be implemented is to perform each maintenance operation twice with interrupts disabled. By doing the operation twice in back-to-back successions with no other memory operations executed

in between, it can be assured that the line will be evicted from both the L1 and L2 cache and written out to main memory.

1. disable interrupts and imprecise aborts
2. execute maintenance operation first pass
3. execute same maintenance operation, second pass
4. enable interrupts and imprecise aborts

Repeat the above sequence for each cache maintenance operation. Interrupts can remain disabled for a longer sequence of maintenance operations, but this will have a negative effect on interrupt latency. This workaround will have a performance impact on the execution time of cache maintenance operations.

586323: Cache clean memory ops generated by either the Preload Engine or Clean by MVA to PoC instructions may corrupt memory

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p2,r2p5,r2p6, Fixed in r2p3,r3p0

Description

When a Clean to Point of Coherency by MVA operation is done, or the Preload Engine is programmed to clean a region of memory from the L2 cache, it is possible that a cache line from that region could be corrupted with a stale copy of memory and a memory store operation will be lost.

NOTE: This erratum corresponds with bug #5072 in the ARM internal bugzilla database.

Conditions

1. A Cache Clean by MVA to the Point of Coherency instruction is executed to clean cache line A, or the preload engine is configured to clean a memory region which includes cache line A. Either operation results in the placement of cache line A into a victim buffer for writeback to external memory, but also keeps the line still valid in the L2 cache.
2. A memory store operation is done to the same cache line A that has been evicted by the cache clean operation. This operation results in a modification of cache line A in the L2 cache (but not to the copy of the line that may still remain in the victim buffer if memory response is slow).
3. A cache eviction is done of cache line A due to an unrelated memory request to load cache line B. The modified copy of cache line A is placed in a victim buffer.

At this point, it is possible that two victim buffers can contain two different versions of cache line A. Since each victim buffer uses a different AXI ID and arbitrates independently for the AXI bus, there is no guaranteeing which order the memory updates will occur, and the store operation may be overwritten by the cache clean operation leaving external memory with stale contents.

Implications

If the operation sequence occurs as described above, one or more store operations will be lost, resulting in incorrect program behaviour. This could occur for any application which either makes use of the preload engine to clean a memory region or uses Clean by MVA to Point of Coherency maintenance operations to clean a region of memory.

Workaround

There are two feasible workarounds that can be used for this erratum.

The first workaround is to place a DMB or DSB barrier at the end of each cache clean routine or preload engine memory clean sequence. This barrier operation is to ensure that the cleaned line will go out and be seen by main memory before the store is executed and therefore guarantee that the clean is done correctly and memory contains the correct final value. This workaround is consistent with the ARM recommended practice for how a maintenance routine should end.

The above workaround is convenient to implement and should work for all expected usage models. However, there is still the possibility that an interrupt could be taken during the clean routine, and that the interrupt handler could do a store to the line just cleaned, allowing for the scenario which can lead to the erratum. Therefore another workaround that can be used that avoids even this case is to convert all Clean by MVA to Point of Coherency operations to be instead Clean and Invalidate by MVA to Point of Coherency as shown by the code sequence below.

Replace all uses of:

```
MCR p15, 0, <Rn>, c7, c10, 1   ; Clean Data cache line by MVA to PoC
```

With this instruction:

```
MCR p15, 0, <Rn>, c7, c14, 1   ; Clean and Invalidate cache line by MVA to PoC
```

There is not a Preload Engine equivalent for this second workaround option since it is not possible to configure the preload engine to do a clean and invalidate operation. Therefore, if there are concerns that the DSB based workaround is insufficient, then it would be advisable to simply not use the Preload Engine for cleaning memory regions. The preload engine can be configured to not be accessible at both a user/privilege and non-secure/secure level of granularity. For more information on Preload Engine configurability, please refer to the Cortex-A8 Technical Reference Manual.

586324: Under a specific set of conditions, a cache maintenance operation done by MVA can result in a corruption of memory

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r2p1,r2p2,r2p5,r2p6, Fixed in r2p3,r3p0

Description

If a non-cacheable memory request is subsequently followed by any cache maintenance operation done by MVA, it is possible to end up with corrupted memory.

NOTE: This erratum corresponds with bug #5075 in the ARM internal bugzilla database.

Conditions

1. The level 1 data cache must be a 32k cache
2. The L1 data cache hardware alias checks are enabled (the L1ALIAS bit in the Auxiliary Control Register is set to 0)
3. The virtual memory management used by the operating system does not follow the page coloring guidelines and allows virtual to physical address alias cases to exist on bit [12] of the address
4. A non-cacheable memory request to normal, device, or strongly ordered memory is subsequently followed by a cache maintenance operation done by MVA without any cacheable memory operations executed in between. The non-cacheable memory request can be fully executed, or could be a speculative instruction in the branch shadow that subsequently is flushed.

When the above conditions are met and the cache maintenance operation happens to generate a hash alias scenario on its cache lookup, memory corruption or a false parity error could occur.

Implications

If the operation sequence occurs as described above, then memory can be corrupted or a false parity error can be generated. In addition, even if the workaround as described below is implemented, it is possible that a nonsecure maintenance operation could result in the invalidation of a secure memory location. Therefore, this could possibly be viewed as an avenue for a security attack. However, the contents of secure memory cannot be viewed as a direct result of this erratum and the lack of consistent repeatability would make it very difficult for the user to make use of this erratum as a security attack.

Workaround

First, if full PIPT caching support is not required by the operating system, or the processor includes a 16k L1 data cache, then no workaround is required.

If alias conditions can occur, then the workaround is to guarantee that a cache maintenance operation is not immediately preceded by a non-cacheable memory request. This can be guaranteed by initiating every cache maintenance by MVA routine with a cacheable load or store request immediately preceding the main loop and ending with a DSB barrier operation at the end of the loop. The load or store that precedes the loop can be done to any cacheable memory location. In addition, both interrupts and aborts will need to be masked during the cache maintenance routine. Interrupt masking is required to prevent a non-cacheable memory request, either fully executed or in a branch shadow, from initiating the sequence that can result in this erratum. If there are

concerns about interrupt latency, the maintenance loop could be amended to do the enable and disable of interrupts directly around the maintenance operation. This will of course impact the time taken to complete the maintenance loop.

To work around any concerns of a potential security attack due to this erratum, all secure memory should be marked as inner writethrough. This can be done either by using the caching attributes in the page tables for all secure page tables or by making use of the secure banked version of the remap registers. In addition to making all secure memory write through, a routine should be run out of reset to completely fill the cache with dummy data to prevent invalid, uninitialized data in the cache from being written out to memory and potentially corrupting secure memory. Making all secure memory inner writethrough guarantees that even if the invalidation of a secure line in the L1 cache occurs due to this erratum, the correct data will not be lost.

621766: Under a specific set of conditions, executing a sequence of NEON or vfp load instructions can cause processor deadlock

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

When a sequence on Neon load instructions is intermixed with several branches, some of which are mispredicted, it is possible for the processor to deadlock due to Neon loads on the speculative path not being properly flushed from the Neon load queue.

NOTE: This erratum corresponds with bug #5095 in the ARM internal bugzilla database.

Conditions

1. Neon or vfp must be enabled
2. The L1NEON bit in the Auxiliary Control Register must be disabled
3. A sequence of VLD instructions must be executed intermixed with a sequence of branches, some of which are mispredicted

If the above conditions are met, it is possible for the Neon load queue to incorrectly update the entry allocation pointer. Once this queue state is reached, the processor will eventually deadlock since the queue contains an entry that can never be de-allocated.

Implications

If this erratum is encountered and processor deadlock occurs, it can only be interrupted by pulling the RESET pin on the processor.

Workaround

The workaround for this erratum is to force all NEON loads to be cached in the L1 cache by setting the L1NEON bit (bit 5) in the CP15 Auxiliary Control Register. This can be done with the following code sequence:

```
MRC p15, 0, R1, c1, c0, 1    ; read Aux Ctl Register
ORR R1, R1 #(1 << 5)         ; set L1NEON to 1
MCR p15, 0, R1, c1, c0, 1    ; write Aux Ctl Register
```

The above code needs to be executed in Secure state. ARM Limited recommends that this code be added to the boot monitor, making the workaround completely transparent to all applications.

This workaround will have a minor performance impact on some Neon and vfp applications since setting the L1NEON bit disables the L2 streaming feature for Neon and vfp data. The measured performance impact of this for Neon applications has been on the order of a few percent (2-3%), but could be more than that for an application that makes heavy use of memory streaming. For such applications PLD instructions could be used to mitigate the performance impact.

621816: Clean and Clean/Invalidate maintenance ops by MVA to PoC can result in processor deadlock

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p3, Fixed in r3p0.

Description

When a Clean to Point of Coherency or Clean and Invalidate to Point of Coherency by MVA operation is done to a cache line present in the L1 cache and not present in the L2 cache, it is possible that the operation will only partially complete and the processor will deadlock.

NOTE: This erratum corresponds with bug #5096 in the ARM internal bugzilla database.

Conditions

1. The memory region being cleaned is configured in Write Allocate mode
2. The cache line being cleaned is initially present in the L1 cache and not the L2 cache
3. The cache line being cleaned has at least 2 different quadwords that contain dirty data
4. The Tag RAM is configured to have a 3 cycle or higher latency - or - an L2 request from the Fetch unit occurs while the cache maintenance operation is taking place

Implications

Having a L2 Tag RAM latency of 2 cycles greatly reduces the likelihood of hitting this erratum. However, while rare, the fetch unit can still make a request at any time, including during the maintenance operation. So, the erratum can still occur even in systems with a 2 cycle Tag RAM latency. If this erratum is encountered and processor deadlock occurs, it can only be interrupted by pulling the RESET pin on the processor.

Workaround

The first workaround for this erratum is to disable Write Allocate in the Level 2 cache. This can be done with the following instruction sequence:

```
MRC p15, 1, <Rd>, c9, c0, 2      ; read L2 cache Aux Ctrl Reg
ORR <Rd>, <Rd>, #(1 << 22)       ; set the Write Allocate disable bit
MCR p15, 1, <Rd>, c9, c0, 2      ; write the L2 cache Aux Ctrl Reg
```

This workaround applies for all configurations of L2 Tag RAM latency. Disabling write allocate in the Level 2 cache could have a performance impact for some applications.

The second workaround for this erratum is to enable bit 17 in the Auxiliary Control Register. This can be done with the following instruction sequence:

```
MRC p15, 1, <Rd>, c1, c0, 1      ; read the Aux Ctrl Reg
ORR <Rd>, <Rd>, #(1 << 17)       ; set the CP14/15 IDLE required bit
MCR p15, 1, <Rd>, c1, c0, 1      ; write the Aux Ctrl Reg
```

This workaround only applies for systems with an L2 Tag RAM latency of 2 cycles. This workaround also has a performance implication on the execution of instructions that access the following cp15 registers.

CRn	Op1	CRm	Op2	Operation
c1	0	c0	2	Coprocessor Access Control
c5	0	c0	1	Instruction Fault Status
c5	0	c1	1	Instruction Auxiliary Fault Status
c5	0	c0	0	Data Fault Status
c5	0	c1	0	Data Auxiliary FaultStatus
c6	0	c0	0	Data Fault Address
c6	0	c0	2	Instruction Fault Address
c7	0	c4	0	PA
c7	0	c5	0	Invalidate All Instruction Caches To PoU
c7	0	c5	4	Flush Prefetch Buffer
c7	0	c5	6	NOP (Invalidate Entire Branch Predictor Array)
c7	0	c5	7	NOP (Invalidate Branch Predictor Array Line By MVA)
c7	0	c6	1	Invalidate Data Cache Line To PoC By MVA
c7	0	c6	2	Invalidate Data Cache Line By Set and Way
c7	0	c8	0-3	VA to PA translation in the current world
c7	0	c10	1	Clean data cache line to PoC by MVA
c7	0	c10	2	Clean data cache line by set and way
c7	0	c10	4	Data Synchronization Barrier
c7	0	c10	5	Data Memory Barrier
c7	0	c11	1	Clean data cache line to PoU by MVA
c7	0	c14	1	Clean and invalidate data cache line to PoC by MVA
c7	0	c14	2	Clean and invalidate data cache line by set and way
c8	0	c5	0	Invalidate Instruction TLB unlocked entries
c8	0	c5	1	Invalidate Instruction TLB entry by MVA
c8	0	c5	2	Invalidate Instruction TLB entry on ASID match
c8	0	c6	0	Invalidate Data TLB unlocked entries
c8	0	c6	1	Invalidate Data TLB entry by MVA
c8	0	c6	2	Invalidate Data TLB entry on ASID match
c11	0	c0	0-3	PLE Identification and Status
c11	0	c5	0	PLE Internal Start Address
c11	0	c7	0	PLE Internal End Address
c11	0	c8	0	PLE Channel Status
c12	0	c0	0	Secure or Nonsecure Vector Base Address
c12	0	c0	1	Monitor Vector Base Address

The large majority of these instructions are executed infrequently and none of them are found in application code. The only commonly executed instructions from this list would be the Cache and TLB maintenance operations. Therefore, it is in maintenance routines that the performance difference might be noticed. ARM has measured a maximum performance delta of 20% for these routines in a worst case system, but actual performance will vary based on system design and the impact could be much less than that depending on the system design.

628216: If a Perf Counter OVFL occurs simultaneously with an update to a CP14 or CP15 register, the OVFL status can be lost.

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p1,r1p2,r1p3,r1p7,r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1,r3p2, Open.

Description

If the PMU is in use and an overflow event occurs simultaneously with a write to one of a subset of CP15 and CP14 registers, it is possible for the overflow event to be lost and, on r1pX revisions of the design, it is also possible that all the PMU registers are corrupted.

NOTE: This erratum corresponds to bug #5097 in the ARM internal bugzilla database.

Conditions

1. The performance counters must be in use
2. The performance counter must have an overflow (counter value goes beyond 0xFFFF_FFFF)
3. Simultaneous with the counter overflow, a MCR instruction must be executed that writes to one of the following CP14/CP15 registers:

- any PMU register other than PMU counter registers
- ThumbEE Configuration Register
- ThumbEE Handler Base Register
- System Control Register
- Auxiliary Control Register
- Secure Configuration Register
- Secure Debug Enable Register
- Nonsecure Access Control Register
- Context ID and Thread ID Registers
- Coprocessor Access Register
- Cache Size Select Register

Implications

If the erratum occurs, the overflow status flag is not set for that counter in the Overflow Flag Status Register, and an interrupt request is not generated, even when the Interrupt Enable Set Register is configured to generate an interrupt on counter overflow. In r1pX revisions of the design, in addition to losing the overflow status for the effected counter, PMU register state may be corrupted for all PMU registers.

Workaround

The main workaround for this erratum is to poll the performance counter. The maximum increment in a single cycle for a given event is 2. Therefore, polling can be infrequent as no counter can increment by more than 2^32 in fewer than 2 billion cycles.

If the main usage model for performance counters is collecting values over a long period, then polling can be used to collect values (and reset the counter) rather than waiting for an overflow to occur. Polling can be done infrequently and overflow avoided. This workaround is valid for both r1 and r2 versions of the design.

If the main usage model for performance counters relies on presetting the counter to some value and waiting for an overflow to occur, then polling can be used to detect when an overflow event has been missed. An overflow can be determined to have been missed if the unsigned value in the counter is less than the value preset into the counter. Again, polling can be done infrequently because of the number of cycles it would need for this check to fail. In the case that the erratum was triggered and an overflow event was missed, that counter sample can be thrown away or the true value can be reconstructed. This workaround is valid only for r2pX and higher revisions of the design, as in r1pX revisions the failure mechanism is more significant since all PMU state is lost.

709718: Load and store operations to shared device memory regions may not complete in program order

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1 Fixed in r3p2.

Description

If a sequence of loads and stores are done to a memory region that is marked as shared device, it is possible for a load to incorrectly bypass a store if the load and store are done to different addresses.

NOTE: This erratum corresponds to bug #5115 in the ARM bugzilla database.

Conditions

1. One or more memory regions are marked as shared device in the memory map
2. A store operation is done to an address ‘A’ that falls within a shared device memory region
3. A load operation is done to a different address ‘B’ that falls within the same shared device memory region (as defined by the system implementation)

Implications

If the load address and store address are mapped to access the same device memory region, and that device relies on memory operations to occur in program order, then this device may not work as intended.

Workaround

The erratum only occurs for shared device memory regions and not for non-shared device memory regions. Therefore, this problem can be worked around by using the remap registers to remap all shared device transactions to non-shared device. The only difference between shared device and non-shared device is the AXI ID used for the transactions on the AXI interface. Shared transactions are broadcast with an AXI ID value of 0x1, non-shared device transactions are broadcast with an AXI ID value of 0x3. Therefore, the user would not experience any impact in terms of performance from this workaround. Another possible use of TEX remap would be to map shared device regions to strongly ordered. This second remapping option is less desirable since performance would be effected as strongly ordered transactions are not buffered.

Here is the code sequence needed to setup and enable TEX remap. This should be done before enabling the MMU.

```
; Setup PRRR so device is always mapped to non-shared
    MRC p15, 0, r0, c10, c2, 0   ; Read Primary Region Remap Register
    BIC r0,#3<<16
    MCR p15, 0, r0, c10, c2, 0   ; Write Primary Region Remap Register
; Enable TEX remap
    MRC p15, 0, r0, c1, c0, 0    ; Read Control Register
    ORR r0,r0,#1<<28
    MCR p15, 0, r0, c1, c0, 0    ; Write Control Register
```

Another valid workaround is to place a data memory barrier (DMB) between all memory accesses to device regions where ordering is required between a store and a subsequent load to a different physical address.

725233: PLD instructions executed with PLD data forwarding enabled can result in a processor deadlock

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: r1p0,r1p1,r1p2,r1p3,r1p7 Fixed in r2p1.

Description

If a PLD instruction is executed and causes a linefill to occur to the L2 cache, and a subsequent Neon or VFP load instruction is done to an address that falls within the same cache line, it is possible for processor deadlock to occur. The deadlock occurs only in a particular timing corner case that can happen when the VLD instruction execution occurs prior to the PLD data being returned on the AXI bus, and there is a Neon resource stall preventing instruction issue during the same time window.

Conditions

1. VFP and Neon are enabled and in use
2. PLD operations are enabled and used with data forwarding enabled
3. A PLD is executed and causes a cache line fill to occur to the L2 cache
4. A subsequent VLD instruction is executed requesting data from the same cache line as the PLD
5. The execution of the VLD instruction occurs prior to the data appearing on the AXI bus, enabling the use of the PLD direct data forwarding path
6. The VLD instruction is delayed from issuing for a few cycles due to a Neon/VFP resource stall condition
7. A handshake is missing between the PLD direct data forwarding path and the issue stall condition, and the data is dropped, resulting in deadlock

Implications

If the deadlock state occurs, it can only be exited by asserting the RESET pin on the processor.

Workaround

There is a simple and straightforward workaround for this issue which is to disable the PLD data forwarding feature, bit [27] in the L2 Auxiliary Control Register. This can be done using the following code sequence:

```
MRC p15, 1, r0, c9, c0, 2   ; read register
ORR r0, r0, #(1<<27)        ; L2 PLD data forwarding disable
MCR p15, 1, r0, c9, c0, 2   ; write register
```

The PLD data forwarding path was added in the design with the intent of reducing the latency of PLD operations in the case where the load requiring the data is executed prior to the PLD data being returned. However, the additional latency savings are small, and the frequency of this occurring in actual practice is rare. Therefore, the additional performance seen from the use of this PLD forwarding feature in practice is negligible and the use of this workaround is not expected to result in any measurable degradation in performance (or power consumption).

728018: Cache maintenance operations by MVA for a non-cacheable memory region can result in processor deadlock

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 2, Present in: all versions, Open.

Description

If a Clean by MVA, Invalidate by MVA, or Clean and Invalidate by MVA cache maintenance operation is done to a region of memory that is marked non-cacheable, device, or strongly ordered, it is possible for the processor to deadlock or have stale data left in the processor. For the erratum to occur, the address must hit in the cache, and it must hit in a different way of the cache than what is predicted by the Hash Virtual Address Buffer (HVAB), which is a cache way predictor inside the processor. This erratum can only occur for the cache maintenance operations that are done by MVA. It will not occur for the set/way based cache maintenance operations.

Conditions

1. A memory region is marked cacheable in a page table entry, and a cache line from that region is placed in the data cache
2. A second page table entry marks the same memory region as non-cacheable, device, or strongly ordered. This could occur by changing the memory attributes in the existing page table entry or through an alternative page table entry that maps the same virtual to physical address but with non-cacheable, device, or strongly ordered attributes rather than cacheable
3. A Clean by MVA, Invalidate by MVA, or Clean and Invalidate by MVA cache maintenance operation is done to this address
4. The maintenance operation gets a false hit indication from the HVAB array
5. The maintenance operation gets a true hit indication from the Tag lookup, meaning the data is present in the array, but it is located in a different way that what was predicted by the HVAB
6. An eviction of the dirty line is started but is not finished and the processor leaves stale data in the cache and can potentially enter a deadlock state

Implications

If stale data is left in the cache, the processor will not work as intended. If the deadlock state occurs, it can only be exited by asserting the RESET pin on the processor.

Workaround

There are two possible workarounds for this erratum.

The first workaround is to avoid doing cache maintenance operations to non-cacheable addresses that had previously been marked cacheable and therefore might be resident in the cache. If the address is found in the cache, this means the memory region must have been marked cacheable at some earlier point in time and explicitly changed to non-cacheable before the maintenance operation is executed. If the region type is not changed to non-cacheable before executing the maintenance operation, the erratum can be avoided. The value of changing a memory region from cacheable to non-cacheable before doing the maintenance operations is that this is the only way that the ARM v7 Architecture guarantees the line will not be immediately placed back into

the cache due to the possibility of data speculation. However, in Cortex A8, this degree of data speculation is never done. Therefore, changing the memory type to non-cacheable before executing the cache maintenance operation is not required for guaranteeing the line is not immediately placed back into the cache. However, if code compatibility with other v7 implementations that might exhibit this level of data speculation is a concern, then this first workaround is insufficient, and the second workaround should be used.

The second workaround is to execute the loop of cache maintenance operations twice. Execute the loop once with the memory region still marked cacheable. Then change the page table entry to make the memory region non-cacheable and execute the loop for a second time. The first loop will clean the data from the cache in the Cortex A8. On the Cortex A8, the second loop is redundant since it will miss on all lines in the cache, but it will resolve the data speculation issue that could be possible on a different v7 architecture implementation. Note that existing cache maintenance code in a dynamically paged environment might be dependent on the maintenance operation triggering a page fault to set the correct page table entry. The workaround code must independently ensure that the correct page table entry is present.

Errata - Category 3

424735: Preload engine transfers can delay draining of write buffer

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

When the preload engine is in use and ECC error correction is enabled in the L2 cache, it is possible under specific corner case conditions for the preload engine transfers to be processed at higher priority than pending store operations in the write buffer. This is incorrect because the preload engine should always be the lowest priority request in the level 2 cache request arbitration.

NOTE: This erratum corresponds to bug #4666 in the ARM internal Bugzilla database

Conditions

1. The preload engine is enabled and is currently in use
2. ECC error correction is enabled in the L2 cache

Implications

If the erratum occurs, store traffic will be stalled in the write buffer until the pending preload engine transfer has completed. This could mean that the store buffer, and subsequently the processor, could be stalled from further execution in the worst case for several thousand cycles. Such a stall would be extremely rare, and could only occur once for each use of the preload engine.

Workaround

If ECC logic is enabled in the L2 cache and the preload engine is used, make sure preload operations have completed before executing any performance critical code sequences. Otherwise, the code could take longer to complete than expected.

430312: A small portion of the ARM opcode space is erroneously decoded as PLI rather than UNDEFINED

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

32-bit ARM opcodes of the form: ARMopcode[31:0] = 1111_0110_x101_xxxx__1111_xxxx_xxx1_xxxx (where "x" is "don't-care") are erroneously decoded as PLI instructions rather than undefined opcodes. The erratum exists because the instruction decoder ignores all bits [11:0] when classifying this combination of opcode bits as a valid PLI instruction. Bit[25] being set to 1 with bit[4] being set to 1 causes this combination of opcode bits to fall outside of the specification of a valid register addressing mode. Therefore, the opcode should fall into the UNDEFINED space and not be executed as a PLI instruction.

NOTE: The erratum corresponds to bug #4771 in the ARM internal bugzilla database.

Conditions

1. A ARM opcode matching the format specified must be manually created in a code sequence
2. An ARM opcode of the format specified above must be executed.

Implications

It is not likely that this erratum will ever be seen by a user since the opcode required for this erratum does not map to any existing ARM instruction. To generate this opcode, the user would have to explicitly create the opcode manually using a data directive (DCI in the ARM assembler). Since the Cortex A8 implementation of the PLI instruction is a NOP (no operation), execution of this opcode would still not result in any visible effect on the code sequence.

Workaround

There is no workaround planned for this erratum.

434613: Incorrect fault status reported for small pages with the XN attribute set

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

A page table lookup that translates a small page with the XN (execute never) attribute set will report the incorrect fault status. A translation fault will be reported, rather than an access flag fault. This erratum occurs for table lookups that occur due to TLB misses on both instruction and data accesses, as well as PLE accesses.

Conditions

1. The AP[0] bit in the translation tables must be configured as the access flag by setting the System Control Register AFE bit [29] to a 1.
2. The translation table must contain at least one small (4 KB) page with the XN attribute set.
3. A table walk must occur to this page, either due to an instruction fetch, load/store, or PLE access.

NOTE: This erratum corresponds to bug #4802 in the ARM internal bugzilla database.

Implications

If the table walk was due to an instruction fetch or CP15 Invalidate I-Cache by MVA operation, the Instruction Fault Status Register (IFSR) fault status field will be incorrect. If the table walk was due to a load/store operation or CP15 data cache operation, the Data Fault Status Register (DFSR) fault status field will be incorrect. If the table walk was due to a preload engine operation, the error code in the PLE Channel Status Register will be incorrect.

In all three cases, the fault status field will indicate "Translation Fault - Page" (encoding 0x7) rather than the correct "Access Flag Fault - Page" (encoding 0x6) value. The abort handler cannot use the fault status register to definitively distinguish between these two fault types.

Workaround

The operating system prefetch and data abort handler code will often first read the fault status register to determine what course of action to take. The workaround involves checking the lower 2 bits of the page table entry in the page translation fault handler to determine if the translation fault is real or is a misdirected access flag fault. Often, the handler code will need to do a software table walk in order to get the address of the page table entry that caused the fault. A true page translation fault occurs when the two -bit descriptor type field (bits [1:0]) of the page table entry is zero; therefore, any other value must really be an access flag fault. Although the incorrect translation fault status value will only occur for a small page with the XN bit set (bits [1:0] == 3),) a translation fault will not occur for a nonzero descriptor type, so both bits need not be compared. If the descriptor type indicates a valid page, then the handler should jump to the page access fault handling code.

Here is annotated assembly to illustrate the workaround:


```
Data_Abort_Handler:

          <save state>
          MCR   p15, 0, r0, c5, c0, 0       ; read the DFSR into R0
          CMP   r0, #7                      ; translation fault on page?
          BEQ   XLat_Fault_Page             ; if so, handle it
XLat_Fault_Page:
          <do software table walk - read page table entry into R0>
          TST   r0, #2                      ; check for bit[1] = 1
          BNE   AFlagFault_Page_Handler     ; if so, handle as page access
                                            ; flag fault
```


436567: CP15 Instruction Set Attribute Register 4, bits 23:20, read 0x1

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

Cortex-A8 implements the following synchronization primitives:

1. LDREXD, LDREX, LDREXH, LDREXB
2. STREXD, STREX, STREXH, STREXB
3. CLREX

According to the ARM Architecture Reference Manual, an ARM processor that implements this set of synchronization primitives must read 0x2 for bits 15:12 of the CP15 Instruction Set Attribute Register 3, and 0x0 for bits 23:20 of the CP15 Instruction Set Attribute Register 4. However, Cortex-A8 incorrectly reads 0x1 for bits 23:20 of the CP15 Instruction Set Attribute Register 4.

NOTE: This erratum corresponds to bug #4825 in the ARM internal bugzilla database.

Implications

0x1 is an invalid value for bits 23:20 of the CP15 Instruction Set Attribute Register 4. This invalid value may cause Operating System malfunction.

Workaround

The Operating System should not rely on the CP15 Instruction Set Attribute Register 4 value in order to identify the set of synchronization primitives that are available. It should use some other means instead like, for example, reading the part number from the CP15 Main ID Register.

436613: ETM OS save sequence resets ETM state

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

The Operating System (OS) Save/Restore registers in the ETM are used to save the state of the ETM before it is powered down, and to restore this state after the ETM is powered back up. While the ETM state is being read out through the OS Save/Restore registers, or while it is being restored back, a control bit known as the ETM OS Lock must be set. This control bit must be low during normal ETM operation.

The ETM architecture specifies that the OS save sequence must not have any side-effects. That is, the ETM state must not be changed by this save sequence. However, Cortex-A8 incorrectly resets this state when the ETM OS Lock is cleared after the save sequence:

1. Counters
2. Sequencer
3. 'Triggered' bit in the ETM Status register
4. TraceEnable Start/Stop block
5. Instrumentation Resources

NOTE: This erratum corresponds to bug #4834 in the ARM internal bugzilla database.

Conditions

1. The ETM OS Lock is cleared after ETM OS save sequence

Implications

If the OS goes through the whole process of performing the ETM save sequence, powering down Cortex-A8, powering it back up, and performing the ETM restore sequence, then the ETM state is restored correctly as per the ETM architecture specification and this erratum is not observable.

However, if the OS decides to abandon the attempt at powering down Cortex-A8, the ETM state is left reset as described above. Typically a reason for abandoning would be a higher priority interrupt.

If the ETM state is reset, resources dependent on this state might not behave as expected. For example, an additional Trigger might occur if one had already occurred before the OS save sequence.

Workaround

This is a workaround for Operating System vendors.

If the OS decides to abandon the attempt at powering down Cortex-A8 once it has already started the ETM save sequence, it must still complete the save sequence and perform the whole restore sequence before abandoning and returning to normal operation.

This workaround does not prevent the state of the Instrumentation Resources from being reset, because the ETM architecture does not specify that this state is visible to the ETM OS save and restore sequences.

441933: Incorrect ID register values

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

The following CP15 ID registers return an incorrect value:

1. Processor Feature Register 0, bits 11:8, State2 field: it reads 0x0 where it should read 0x1. The correct value indicates that Cortex-A8 supports the Jazelle extension, which it does. (It supports the so-called trivial implementation of the Jazelle extension.)
2. Memory Model Feature Register 0, bits 31:28: it reads 0x3 where it should read 0x0. This field is reserved.
3. Memory Model Feature Register 3, bits 11:8, Branch Predictor Maintenance: it reads 0x0 where it should read 0x2. The correct value indicates that Cortex-A8 supports all available ARMv7 branch predictor maintenance operations.
4. Instruction Set Attributes Register 1, bits 27:24, Interworking instructions: it reads 0x2 where it should read 0x3. The correct value indicates that Cortex-A8 supports all ARM/Thumb interworking instructions.
5. MPU Type Register: this register reads 0x0 where it should be an alias of the Main ID Register.

The following debug ID registers return an incorrect value:

1. Debug register at address 0xD10 reads 0x0 where it should read the same value as the Main ID Register
2. Debug register at address 0xD18 reads 0x0 where it should read the same value as the Main ID Register
3. Debug register at address 0xD1C reads 0x0 where it should read the same value as the Main ID Register

The CP15 Secure Configuration Register, bit 6, Early Termination Disable control bit can be written to a 1 but has no effect. The correct behavior for this bit is that it should always read 0.

NOTE: This erratum corresponds to bug #4856 in the ARM bugzilla database.

Implications

Any piece of software that relies on correct values for these ID registers may malfunction.

Workaround

There is not a workaround available for this erratum.

441934: TLB page table walks and Neon load/stores pollute the cache while in debug state

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If Cortex-A8 is in debug state and bit 0, nDL, of the Debug State Cache Control Register is set to 0 then memory operations should not cause line fills or line evictions. However, if a Neon load/store or TLB page tablewalk occurs while in debug state with the nDL bit cleared, the L1 data cache and/or L2 unified cache may be updated.

Conditions:

1. Cortex-A8 in debug state
2. nDL bit set to 0
3. The debugger executes a Neon load/store, or a TLB page tablewalk occurs due to debugger executing any type of memory operation that misses in the TLB.

NOTE: This erratum corresponds to bug #4885 in the ARM bugzilla database.

Implications

If, while Cortex-A8 is in debug state, the debugger executes a Neon load/store instruction or generates a TLB page tablewalk, a line may get brought into or evicted from the caches unexpectedly. If this is the case then, when the processor leaves debug state and returns to the application, the caches may be in a different state than they were before entering debug state. This means that the debugging process is not completely transparent to the application. In particular, the time it takes to run the application after returning from debug state may be different depending on the state of the caches at that time.

It is worth pointing out, though, that Cortex-A8 does not always guarantee that it can prevent cache line evictions. The two exception below are documented in the Cortex-A8 Technical Reference Manual:

1. If identical virtual addresses, except for bit [12], are mapped to the same physical address and the line that corresponds to the first virtual address is in the L1 data cache, then an access using the second virtual address causes an eviction of the cache line to the L2 cache.
2. The L1 data cache controller uses a hash algorithm to determine hits. If two different virtual addresses have the same hash and the line that corresponds to the first VA is in the L1 data cache, then an access using the second VA evicts the line to the L2 cache.

Workaround

Instead of executing a Neon load, the debugger could load the value into an ARM register and move this value from the ARM register to the Neon register.

Instead of executing a Neon store, the debugger could move the value from the Neon ARM register to an ARM register and execute an ARM store.

In order to prevent TLB page tablewalks from updating the L2 cache, there are two things the debugger could do:

1. It could turn the MMU off, that is, clear bit 0 of the CP15 System Control Register. There are two problems with this approach: The first one is that this is a privileged instruction that the debugger will not be able to execute while in debug state if the processor is in Secure-User mode and the SPIDEN input is low. The second one is that when the MMU is turned off memory operations behave differently. In particular, there is a flat address mapping between virtual addresses and physical addresses. If the debuggger does not know what the physical address map is, it may not be able to continue the debug process.
2. It could save the state of the L2 data cache on entering debug state and restoring it on leaving debug state, using the Cortex-A8 CP15 L2 system array debug data registers. The issue with this approach is that these CP15 instructions can only be executed while in a secure privileged mode. The debugger will not be able to get itself into such a mode if the SPIDEN input is low.

451020: A Neon store to device memory can result in dropping a previous store

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If a Neon store is done to device type memory and is followed in instruction sequence by a load instruction that is non-cacheable or cacheable, but misses in the L1 cache, it is possible that a subsequent, unrelated store instruction that is done to cacheable memory will have its data dropped and therefore not update memory.

NOTE: This erratum corresponds to bug #4987 in the ARM internal bugzilla database.

Conditions

1. A Neon store is done to device type memory.
2. Three subsequent ARM stores are done to different addresses that are all cacheable memory type. All three stores hit in the L1 data cache.
3. An ARM load instruction is done that is non-cacheable or cacheable, but misses in the L1 cache.
4. Another store instruction (ARM or Neon) is done to device type memory.

It is one of the three cacheable stores that may not update memory due to this erratum.

Implications

If the erratum occurs, one of three cacheable stores in the described sequence will not update the cache, leaving stale contents in memory. This is likely to cause observable, incorrect behaviour in the application.

Workaround

The only work around for this erratum is to avoid accessing device type memory with Neon store instructions.

451329: A Non-secure invalidate operation can generate an abort response when trying to access a secure line

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

A cache line invalidate operation done to a secure location in the cache should have no effect. However, if a parity or ECC double-bit error is present on a secure line, then the non-secure invalidate operation will generate an abort response.

NOTE: This erratum corresponds to bug #4943 in the ARM internal Bugzilla database.

Conditions

1. Parity/ECC enabled on L2
2. Secure line is present in the L2 cache
3. Non-secure invalidate by set/way is issued targeting the secure line (non-secure process is not allowed to invalidate a secure line)
4. If the secure line contained a parity or double-bit ECC error an imprecise data abort is generated, by the non-secure invalidate operation

Implications

The implications of this erratum are very minor since the abort response will only occur if a parity or double-bit ECC error is present on the secure line. Also, the line will not be invalidated. Therefore, there are no security implications from this erratum.

Workaround

There is no known workaround for this erratum.

453297: BKPT on 2nd half of Thumb-2 instruction cause switch to ARM state.

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

A breakpoint that is set at an address including the second half of a 32-bit Thumb-2 instruction but not the first half may cause a spontaneous switch between ARM and Thumb-2 state or vice versa. This is often encountered near a recent execution state change caused by a predictable branch instruction (BLX, BX, or an ALU instruction with PC as the destination). The hardware will only exhibit this erratum if a pipeline flush does not occur after the switch to the new execution state.

NOTE: This erratum corresponds to bug #5021 in the ARM bugzilla database.

Conditions

1. Debug is enabled.
2. Predictable branch instructions are used to switch between ARM and Thumb-2 state, and branch prediction is enabled.
3. A breakpoint is set on a single halfword address corresponding to the second half of a 32-bit Thumb-2 instruction, or a breakpoint is set on an address range that begins with the second half of a 32-bit Thumb-2 instruction, or a mismatch breakpoint is set on the first half of a Thumb-2 instruction.
4. A pipeline flush does not occur between the state switch and the breakpointed address.

Implications

As instructions intended to be executed in Thumb-2 state are executed in ARM state, or vice versa, the processor will interpret the instructions in the wrong processor state. This may lead to an UNDEFINED processor exception or unpredictable, incorrect processor behavior.

Setting breakpoints on address ranges that begin with the second half of a 32-bit Thumb-2 instruction should also be avoided. An example of this case is setting a range breakpoint on range [0x1000, 0x2000) with a 32-bit Thumb-2 instruction located at address 0xFFE. The breakpoint will occur when the address 0x1000 to 0x1FFF is encountered, which corresponds to the conditions for generating the erratum.

Workaround

Do not set a breakpoint on a single halfword address corresponding to the second half of a 32-bit Thumb-2 instruction. The debug architecture specification states that these are IGNORED.

The method of doing single stepping in ARMv7 debug is setting a legal breakpoint on the first half of a 32-bit Thumb-2 instruction. When the breakpoint is hit, the breakpoint handler or debug monitor will change the breakpoint type to a mismatch. This has the effect of setting breakpoints on the second half of the hit instruction, in addition to all other instructions. This will not exhibit the erratum, as hitting the breakpoint will cause a pipeline flush regardless of whether halting or monitor debug mode is used.

468415: Swap instruction, a preload instruction, and an instruction fetch request can interact and cause deadlock.

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r2p1,r2p5, Fixed in r2p2,r2p3,r2p6.

Description

Three memory requests in the L2 cache can interact and result in a deadlock condition. The exact scenario involves a dependency chain of three requests, an instruction fetch request, a memory preload instruction (PLD) and a swap instruction (SWP). In this dependency loop, none of the requests can make progress since each is dependent on the next request. i.e. The PLD request can't complete since the IF request in pending to use the BIU. The IF request can't complete because of the pending SWP request, and the SWP request is not allowed to complete because it is waiting on the PLD to complete before obtaining the lock on the bus.

NOTE: This erratum corresponds to bug #5045 in the ARM Bugzilla database.

Conditions

1. PLD instructions must be used by the processor
2. SWP instructions must be used by the processor

Implications

This erratum will only impact users of swap instructions. Swap instructions have been deprecated from the ARMv7 version of the ARM Architecture since its functional use in terms of setting up semaphores has now been replaced from the ARMv6 architecture forwards by the LDREX and STREX instructions.

If this erratum is encounter and processor deadlock occurs, it can only be interrupted by pulling the RESET pin on the processor.

Workaround

One software workaround for this erratum is to not make use of swap instructions.

If swap instructions must be used in the code base, the other software workaround is to disable PLD instructions, making them a NOP. Here is the code required for implementing this workaround:

```
MRC p15, 0, r0, c1, c0, 1    ; read register
ORR r0, r0, #(1<<9)          ; PLDNOP - force PLD to be NOP
MCR p15, 0, r0, c1, c0, 1    ; write register
```

Implementing this workaround will have some performance impact on peak memory copy bandwidth.

495515: Synchronized entry into debug state from multiple cores in a MP system could result in a debug livelock loop

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7, r2p1,r2p2,r2p3,r2p5,r2p6, Fixed in r3p0.

Description

One of the uses of the Cross Trigger Interface (CTI) CoreSight component is to provide mechanisms for doing multi-core debug for multiprocessor systems. By using the CTI interface, a single event triggered on one processor can cause that processor and all other processors in the system to immediately halt and enter debug state. However, upon subsequent exit from debug state, it is possible to enter a race condition that results in a debug livelock loop where each processor continuously re-enters debug state and will not be able to return to a state of normal program execution.

NOTE: This erratum corresponds to bug #5056 in the ARM Bugzilla database.

Implications

This erratum has no effect on single processor systems or multiple processor systems that do not require synchronous debug for the multiple processors.

For systems that do require synchronous debug of multiple processors, it is possible to hit a debug livelock scenario when synchronous halt and restart is done. If the specified workaround is implemented, this livelock case can be avoided. However, it will not be possible to synchronously re-enter debug state until the CTI register have been remapped through the debug interface. Depending on the system, this could be a significant amount of time (on the order of 100uS to 1mS).

Workaround

This erratum can be avoided by disabling the CTI signal mapping before restarting the processor. This avoids the livelock case of continuously re-entering debug state. However, implementing this workaround also means that synchronous debug entry cannot be done again until the CTI registers are remapped again to re-enable the event trigger. The time it takes to reprogram the CTI registers is system dependent, but could be on the order of 100uS to 1mS.

507113: A Neon store to device memory can result in dropping a previous store

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7,r2p1,r2p2,r2p3,r2p5,r2p6, Fixed in r3p0.

Description

If a Neon store is done to device type memory and is followed in instruction sequence by a load instruction to device type memory, it is possible that an unrelated store instruction that is done to cacheable memory and hit the L1 cache will have its data dropped and therefore not update memory.

NOTE: This erratum corresponds to bug #5057 in the ARM internal bugzilla database.

Conditions

Relatively close in the instruction stream, the following must occur:

1. A Neon store is done to device type memory.
2. A load is executed to device type memory.
3. Several stores hit the L1 cache.

Implications

If the erratum occurs one or more cacheable stores that hit the L1 cache will not update the cache, leaving stale contents in memory. This is likely to cause observable, incorrect behaviour in the application.

Workaround

The only work around for this erratum is to avoid accessing device type memory with Neon store instructions.

588115: A RAW hazard on certain CP15 registers can result in a stale register read

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r2p1,r2p2,r2p3,r2p5,r2p6, Fixed in r3p0.

Description

Under certain conditions, a sequence of instructions where an MCR instruction that writes a CP15 register is closely followed by an MRC that reads the same register may be executed such that the RAW hazard is not detected and the MRC reads the old value of the register.

This scenario can only occur for accesses to a to one of these four CP15 registers that are both readable and writeable and have been optimized to execute in a single cycle:

- CacheSizeSelection Register
- Thread and ProcessID user read/write
- Thread and ProcessID user read only
- Thread and ProcessID privilege only

Furthermore, this scenario can only occur if a specific sequence of instructions is executed between the MCR and the MRC. The sequence must meet two criteria:

1. It must take less than 3 cycles to execute
2. It must have one of the instructions in the following list:

- ARM PLD with [Rn, -Rm, <shift>] addressing mode
- ARM or Thumb PLD with [Rn, Rm,<shift>] addressing mode (unless it is LSL #0 or LSL #2)
- Thumb or ThumbEE load/store instruction with [Rn, Rm,<shift>] addressing mode (unless it is LSL #0 or LSL #2)
- Thumb TBB instruction

NOTE: This erratum corresponds to bug #5079 in the ARM internal bugzilla database.

Conditions

1. Execute an MCR instruction to one of the four registers in the above list, writing a new value
2. Execute an instruction in the second list above
3. Can execute more instructions between the MCR and the following MRC as long as there are less than 3 cycles between the MCR and the MRC
4. Execute an MRC instruction to the same register as the MCR above, incorrectly reading the old value of the register

Implications

If this erratum is encountered, the old stale value of the register will be read rather than the newly written value in which case system software may appear to behave incorrectly. However, the usage model for such a

software sequence in unclear, and hence the likelihood of encountering it in practice seems very low, especially considering the requirement of the 2nd unrelated instruction that must also fall between the MCR and the MRC.

Workaround

If a workaround for this erratum is desired, there are two options. The first option is to simply place a NOP immediately following the MCR register write in any case where hitting this erratum might be a concern. By adding a single NOP, the minimum required cycle window can be guaranteed to be met and the erratum cannot occur. The second option for working around this erratum is to set bit 16 in the CP15 Auxiliary Control Register. This will cause a pipeline flush to occur on every write to a cp15 register and thereby ensure that the RAW hazard condition cannot occur. This second workaround has the advantage of requiring just one change to the cpu configuration that can be done statically, but the disadvantage that it will have some impact on the performance of write updates to cp15 registers that would not otherwise require a pipeline flush.

This second workaround can be implemented with the following code sequence:

```
MRC p15, 0, R1, c1, c0, 1   ; read Aux Ctl Register
ORR R1, R1 #(1 << 16)       ; set bit 16 to 1
MCR p15, 0, R1, c1, c0, 1   ; write Aux Ctl Register
```

The above code needs to be executed in Secure state.

687067: BTB invalidate by MVA operations will not work as intended when the IBE bit is enabled

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7,r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1,r3p2, Open.

Description

All BTB invalidate operations, including BTB Invalidate by MVA operations, by default are implemented as a NOP in the Cortex-A8 processor. These operations can be executed as NOPs since flushing BTB entries is not required by the Cortex-A8 processor for correct functionality, and there is no additional performance penalty for an incorrect branch prediction versus a non-prediction. However, it is possible for BTB operations to be enabled by setting the IBE bit in the CP15 Auxiliary Control Register. When enabled in this fashion, BTB invalidate by MVA operations may not work as intended. Rather than writing 0s to the valid bit of the BTB entry matching the MVA provided, the CP15 "Invalidate Branch Predictor by MVA" operation writes the value currently in the "Instruction L1 System Array Debug Register 0." If the value written into the BTB entry sets the valid bit and clears the page cross bit, it is possible to trigger the conditions described in Defect 657417. This register is not initialized at reset time and can only be written in secure, privileged modes when CP15SDISABLE is not set.

NOTE: This erratum corresponds to bug #5108 in the ARM internal Bugzilla database.

Conditions

1. The branch predictor is enabled (SCTLR.Z = 1)
2. The Auxiliary Control Register IBE bit is set to 1
3. An invalidate Branch predictor by MVA operation is executed
4. The Instruction L1 System Array Debug Register 0 contains a non-zero value which sets the valid bit and clears the page cross bit

Implications

If the non-zero value contained in L1 System Array Debug Register 0 happens to set the valid bit of the BTB entry, then the entry will not be invalidated as intended. If the valid bit is set and the page cross bit is not set, then it is possible that on a subsequent BTB access, a required TLB lookup will not occur, which results in subsequent fetching from incorrect addresses or hangs as described in Defect 657417.

Workaround

One workaround for this erratum is to not enable the IBE bit. ARM recommends that the IBE bit should not be enabled unless it is required for an erratum workaround.

If the IBE needs to be enabled, then the L1 System Array Debug Register 0 should be initialized to a zero value. This register is for RAM array debug purposes and is not used as a part of normal functionality. It is also only accessible in a privileged secure mode. Therefore, it can be statically initialized as a part of the boot code sequence. If the register is used for debug purposes, the value should be reset to zero when the debug sequence completes.

Code to initialize the L1 System Array Debug Register 0 is shown below.

```
MOV r1, #0
MCR p15, 0, r1, c15, c1, 0   ; write instruction data 0 register
MRC p15, 0, R1, c1, c0, 1    ; read Aux Ctl Register
ORR R1, R1 #(1 << 6)         ; set IBE to 1
MCR p15, 0, R1, c1, c0, 1    ; write Aux Ctl Register
```


693270: Taking a watchpoint is incorrectly prioritized over a precise data abort if both occur simultaneously on the same address

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1,r3p2, Open.

Description

If a debug watchpoint and a precise data abort are both triggered from the same data access, the ARM Architecture specifies that the data abort should be prioritized. However, this does not occur on the Cortex A8 and the watchpoint is taken instead.

NOTE: This erratum corresponds to bug #5113 in the ARM internal Bugzilla database

Conditions

1. At least one debug watchpoint is programmed
2. A precise data abort occurs on the same address as the watchpoint

Implications

The implications of this erratum only effect debug software. The data abort should take precedence over the watchpoint so the OS has a chance to fix up paged-out memory before re-executing the instruction and presenting the debugger with the watchpointed address. Due to this erratum, this fix up will not occur and the debugger will need to be capable of handling a faulting address.

Workaround

The workaround for this erratum is making sure the debugger software can handle a faulting address. When the debugger is signalled a watchpoint and discovers the page being accessed is subject to an MMU fault which it would like the OS to patch up before dealing with itself, it can:

- disable the watchpoint
- set vector catch on the local Data Abort exception (secure or non-secure, as appropriate)
- set the PC at the watchpointed instruction and restart execution

The processor will restart, re-execute the instruction and generate the MMU fault. It will then fetch the instruction from the Data Abort handler and re-enter Debug state because of the Vector Catch event. The debugger can then:

- re-enable the watchpoint
- disable the vector catch
- set the PC at the Data Abort vector and restart execution

The processor will restart and re-execute the Data Abort vector instruction. The OS will then patch up the MMU fault and attempt to re-execute the original instruction. Re-executing the instruction will regenerate the Watchpoint debug event, but now the page has been properly patched up.

715847: VCVT.f32.u32 can return the wrong result for the input 0xfff_ff01in one specific configuration of floating point unit

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r2p1,r2p2,r2p3,r2p5,r2p6,r3p0,r3p1,r3p2 Open.

Description

If the integer to floating point conversion operation, VCVT.f32.u32, is executed with the FPSCR register configured for Default NaN and Flush-to-zero enabled, and the rounding mode used is RP, it will return the incorrect result for the source operation 0xffff_ff01. Specifically, it will return the result 0x0000_0000 instead of the correct result 0x4f80_0000. The erratum can only occur for this specific input value and this specific configuration of the FPSCR register.

NOTE: This erratum corresponds to bug #5117 in the ARM bugzilla database.

Conditions

1. Default NaN is enabled (FPSCR[25]=1'b1)
2. Flush-to-zero is enabled (FPSR[24]=1'b1)
3. RP rounding mode is enabled (FPSR[23:22] = 2'b01)
4. A VCVT.f32.u32 instruction is executed with the source operand 0xffff_ff01
5. The result of the instruction will incorrectly be 0x0000_0000 rather than 0x4f80_0000

Implications

The incorrect result from the conversion operation could result in further incorrect results calculated and unexpected program behaviour.

Workaround

The erratum only occurs if the floating point unit is configured in run fast mode with RP rounding (Round-to-positive infinity). The easiest workaround is to avoid using this particular mode combination. Round-to-nearest (RN) is the more common rounding mode used, but if RP functionality is desired, it should be done without using Default NaN and/or without Flush-to-zero enabled. Default NaN signalling, Flush-to-zero, and rounding mode are all configured using bits [25:22] of the FPSCR register. This register is typically configured by system software and should not change within an application.

728021: Load and stores executed in debug state can result in memory corruption when force write through is enabled in the DSCCR

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: Cat 3, Present in: r1p1,r1p2,r1p3,r1p7 Fixed in r2p1.

Description

It is possible to force all write back memory transactions done in debug state to instead be done as write through transactions. This feature is enabled by clearing bit[2] in the DSCCR register. If this feature is enabled, it is possible that dirty data in the L2 cache will not be properly updated to main memory.

Conditions

1. The memory type for a particular region is configured to be WB
2. A cache line from this region (Cache line ‘A’) is dirty in at least one quad word in the L1 cache
3. The same cache line ‘A’ is dirty in a different quad word in the L2 cache
4. The processor enters debug state with the force WT feature enabled
5. Cache line ‘A’ is evicted from the L1 cache while in debug state (note: this will be rare if DSCCR[0]=0 since linefills while in debug state are disabled)
6. The force WT feature causes dirty quad word data from cache line A to be pushed from the L1 cache all the way out to main memory
7. Cache line ‘A’ is updated in the L2 cache
8. The dirty bits in the L2 cache for line ‘A’ are cleared at which point the latest version of this memory is lost and main memory contains stale data

Implications

If memory is not correctly updated, this is likely to result in incorrect program behaviour. Note: This feature is not present in future versions of the ARM architecture.

Workaround

The workaround for this erratum is to ensure the force write through feature in the DSCCR is disabled. This feature is enabled by default and therefore should be explicitly disabled by the debugger as a part of the initialization sequence before debug mode is entered.

The purpose of this feature is to simplify the debugger's job at maintaining memory coherency while the processor is in debug state. In particular, forcing write-through allows the debugger to do these two things without needing to clean the D-Cache: write software breakpoints (BKPT instruction) to memory, and download code to memory through the debug comms channel (DTRRX register). The debugger should instead manually clean the D-Cache using the appropriate cache maintenance operations after manipulating software breakpoints or downloading code.

Errata- Implementation

416257: SAFESHIFTRAM input is not used correctly when doing a partitioned test flow

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: implementation, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

The CORTEXA8 input pin SAFESHIFTRAM is used in test mode to prevent RAM updates during the capture cycle of DFT testing. One part of the intended functionality is to allow the enabling of only the RAMs being tested when a partitioned test flow is used. However, the current implementation will enable all RAMs during the capture cycle even when only enabling a subset of RAMs is required.

NOTE: This erratum corresponds to bug#4554 in the ARM internal Bugzilla database.

Conditions

1. The processor is in test mode, not functional mode (DFT)
2. A partitioned test flow is being used

Implications

The power draw during scan capture may be higher when using a partitioned test flow than it otherwise would be. This may be acceptable depending on the requirements for the DFT methodology used and the specific physical properties of the silicon.

Workaround

If the additional power consumed during partitioned testing is found to be a problem, it can resolved by altering the test flow to use smaller test partitions. This would increase the number of patterns required to achieve full DFT coverage.

Errata – System

451335: ETM may not power up correctly after a power down sequence

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: system, Present in: r1p1,r1p2,r1p3,r1p7, Fixed in r2p1.

Description

If the ETM unit is dynamically powered down and then powered up again using the power up and power down sequences described in the Cortex A8 Technical Reference Manual, it is possible that the ETM will not reset correctly when powered up and end up in an indeterminate state.

NOTE: This erratum corresponds to bug #4844 in the ARM internal Bugzilla database.

Conditions

1. Integer core powered and in normal operation.
2. ATRESETn and PRESETn are asserted (low), placing ETM and DEBUG in reset.
3. Clamps around the DEBUG power domain are activated.
4. The DEBUG power domain is powered down, which includes ETM[CLK], ETM[ATCLK], and DEBUG[PCLK].
5. The DEBUG power domain has power restored (a significant amount of time may have passed).
6. ATRESETn and PRESETn are de-asserted (high), releasing ETM and DEBUG from reset.

At the end of this sequence, The ETM clock domain may not be reset correctly, preventing proper function of the ETM unit.

Implications

If the ETM is dynamically powered down and then powered back up, the trace generated from the ETM after that point may be corrupted.

Workaround

This erratum can be worked around by re-asserting PRESETn for a second time for a pulse of 16 cycles or greater. This will correctly reset the ETM clock domain.

488063: ARPROT[0] is incorrectly set to indicate a USER transaction for memory accesses generated from user tablewalks

Status

Affects: product Cortex-A8 Optimized Implementation.

Fault status: system, Present in: r1p1,r1p2,r2p1,r2p5, Fixed in r1p3,r1p7,r2p2,r2p3,r2p6

Description

All memory transactions done as part of a tablewalk should be considered privileged, even when done while in user mode. However, Cortex A8 will incorrectly mark memory transactions generated from tablewalks done in user mode as user transactions on the AXI bus. This indication is given by the ARPROT[0] signal being set to zero during the transaction.

NOTE: This erratum corresponds to bug #5049 in the ARM internal bugzilla database.

Conditions

1. Cortex-A8 must be in user mode
2. A memory transaction (instruction or data) misses in the TLB and causes a table walk
3. The address for the page table entry is not found in the L2 cache, resulting in an external memory request
4. Erratum occurs here when APROT[0] incorrectly indicates a user transaction for this memory request on the AXI bus

Implications

Since the values broadcast on ARPROT[0] are completely transparent to software, the implications for this erratum are only on a specific subset of processor systems. Specifically, any system that includes some form of system level memory protection unit that is making use of the ARPROT bits to determine if a memory request can be allowed. For any system that does include such a unit, that unit may report false errors on page table accesses due to this erratum.

Workaround

Since the processor does not directly make use of ARPROT[0], any workaround would be specific to the device that was making use of the values broadcast on ARPROT[0]. The most likely usage would be some form of system memory protection unit. If such a protection unit existed, it may need to filter out any access to the page tables from the address space that is protected to work properly. Doing this would mean the external protection unit would not be able to provide additional protection for page tables.

This erratum has been fixed in product revision r1p3 for partners who may be taking the Cortex A8 r1 silicon into production. This r1p3 patch release is not generally available and can only be obtained by contacting an ARM representative.

Errata - Testchip

Currently there are no Testchip Errata.

Appendix A – Product Revision Information

Cortex-A8 genealogy of product revisions:

```
r1p2 => r1p3 => r1p7

r2p1 =>         r2p5  (CAT1 errata fix to r2p1)
r2p1 => r2p2 => r2p6  (CAT1 errata fix to r2p2)
r2p1 => r2p2 => r2p3

r3p0 => r3p1 => r3p2
```

CPUID values of various Cortex-A8 product revisions:

ID value	r1p1	r1p2	r1p3	r1p7	r2p1	r2p2	r2p3	r2p5	r2p6	r3p0	r3p1	r3p2
Main ID Register, Variant field	0x1	0x1	0x1	0x1	0x2	0x2	0x2	0x2	0x2	0x3	0x3	0x3
Main ID Register, Revision field	0x1	0x2	0x3	0x7	0x1	0x2	0x3	0x5	0x6	0x0	0x1	0x2
Debug ID Register, Variant field	0x1	0x1	0x1	0x1	0x2	0x2	0x2	0x2	0x2	0x3	0x3	0x3
Debug ID Register, Revision field	0x1	0x2	0x3	0x7	0x1	0x2	0x3	0x5	0x6	0x0	0x1	0x2
Peripheral ID2 Register, Revision field	0x2	0x2	0x2	0x2	0x4	0x5	0x7	0x4	0x5	0x6	0x6	0x6
Peripheral ID3 Register, RevAnd field	0x0	0x1	0x2	0x3	0x0	0x0	0x0	0x1	0x1	0x0	0x1	0x2
FPSID Register, Revision field	0x1	0x1	0x1	0x1	0x2	0x2	0x2	0x2	0x2	0x3	0x3	0x3

