ARM Processor Cortex™-A5 MPCore™

Product Revision r0

Software Developers Errata Notice

Non-Confidential - Released

Software Developers Errata Notice

Non-Confidential Proprietary Notice

Web Address

http://www.arm.com

Feedback on content

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

- the document title
- the document number, ARM-EPM-025300
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Release Information

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

03 Oct 2012: Changes in Document v1

Page	Status	ID	Cat	Rare	Summary of Erratum
8	New	741950	CatA	Rare	Data hazards between non-NEON FP and certain NEON instructions can deadlock the processor under certain conditions
9	New	745469	CatB		Data prefetcher can cause a deadlock when shared and non-shared data is resident in the data cache
10	New	761273	CatB		Shareable cacheable write data may not automatically become visible to ACP
10	New	743633	CatB	Rare	Memory requests from out-of-date page mappings may be observable after a forwarded TLB invalidate operation is executed
12	New	744929	CatB	Rare	Unexpected instruction prefetch abort can occur in privileged modes
13	New	753869	CatB	Rare	VFPv4 and NEON fused multiply accumulate result may be incorrectly signed when addend is infinite
14	New	753870	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded
17	New	761269	CatB	Rare	Exclusive monitor might incorrectly remain set in one CPU core when another has exclusive access
19	New	765872	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded
22	New	765873	CatB	Rare	VFPv4 fused multiply accumulate result may be incorrectly signed when addend is infinite
23	New	769221	CatB	Rare	Stores to out-of-date translation table mappings might be observable after a TLB invalidate operation completes
24	New	745769	CatC		Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding
25	New	753670	CatC		VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode
26	New	765870	CatC		A load from normal memory marked with a debug watchpoint may result in an external abort
27	New	765871	CatC		Watchpointed access in a store-multiple is not masked
28	New	765874	CatC		VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode
29	New	767976	CatC		Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes
31	New	781769	CatC		ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation
32	New	791671	CatC		A Non-Cacheable store in a tight loop might not become observable until the loop completes

Contents

Chapter 1.
Introduction

This chapter introduces the errata notice for the ARM Cortex-A5 MPCore processor.

1.1. Scope of this document

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

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

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

1.2. Categorization of errata

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

Table 1 Categorization of errata

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

Chapter 2.
Errata Descriptions

2.1. Product Revision Status

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

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

p*n* Identifies the minor revision or modification status of the product.

2.2. Revisions Affected

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

This document includes errata that affect revision r0 only.

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

Table 2 Revisions Affected

ID	Cat	Rare	Summary of Erratum	r0p0	r0p1
741950	CatA	Rare	Data hazards between non-NEON FP and certain NEON instructions can deadlock the processor under certain conditions	X	
761273	CatB		Shareable cacheable write data may not automatically become visible to ACP	X	X
745469	CatB		Data prefetcher can cause a deadlock when shared and non-shared data is resident in the data cache	X	
769221	CatB	Rare	Stores to out-of-date translation table mappings might be observable after a TLB invalidate operation completes	X	X
765873	CatB	Rare	VFPv4 fused multiply accumulate result may be incorrectly signed when addend is infinite	X	X
765872	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded	X	X
761269	CatB	Rare	Exclusive monitor might incorrectly remain set in one CPU core when another has exclusive access	X	X
753870	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded	X	X
753869	CatB	Rare	VFPv4 and NEON fused multiply accumulate result may be incorrectly signed when addend is infinite	X	X
744929	CatB	Rare	Unexpected instruction prefetch abort can occur in privileged modes	X	
743633	CatB	Rare	Memory requests from out-of-date page mappings may be observable after a forwarded TLB invalidate operation is executed	X	
791671	CatC		A Non-Cacheable store in a tight loop might not become observable until the loop completes	X	X
781769	CatC		ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation	X	X

ID	Cat	Rare	Summary of Erratum	r0p0	r0p1
767976	CatC		Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes	X	X
765874	CatC		VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	X	X
765871	CatC		Watchpointed access in a store-multiple is not masked	X	X
765870	CatC		A load from normal memory marked with a debug watchpoint may result in an external abort	X	X
753670	CatC		VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	X	X
745769	CatC		Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding	X	

2.3. Category A

There are no errata in this category

2.4. Category A (Rare)

741950: Data hazards between non-NEON FP and certain NEON instructions can deadlock the processor under certain conditions

Category A Rare

Products Affected: Cortex-A5 NEON Media Engine.

Present in: r0p0

Description

Under certain circumstances, when one of a subset of NEON instructions is executed after certain non-NEON floating point instructions it can either produce the wrong result or cause a deadlock in the processor. This errant behaviour is caused by an error in the logic controlling the data dependency between two or more operations in the processor pipeline in circumstances where one can complete out-of-order with respect to the generated code stream.

Conditions

For this erratum to occur, one of the following code sequences must be executed in either ARM or Thumb state:

Case 1

1) one of the following 64-bit floating point multiply/fused multiply instructions: VMUL.F64, VFMA.64, VFNMA.F64, VFMS.F64, VFNMS.F64
2) one of the following subset of integer NEON instructions: VADDHN, VADDW, VRADDHN, VRSUBHN, VSUBHN, VSUBW, VUZP, VZIP, that has a RAW hazard with 1.

Case 2

1) VSQRT.F32 or VDIV.F32
2) a VCVT.F32.F16 instruction which has overlapping source and destination operands and which has a WAW dependency on 1.

Case 3

1) VSQRT.F32 or VDIV.F32
2) VEXT that has a RAW hazard with 1. and:

- the immediate of the VEXT is < 8
- the destination register of the VEXT is also used as the second source register
- the VDIV.F32 / VSQRT.F32 is updating S<4n+3> - i.e. the register which maps onto the most significant word of Q<n>

Case 4

1) one of: VSQRT.F32, VSQRT.F64, VDIV.F32 or VDIV.F64
2) one of: VMUL.F64, VFMA.64, VFNMA.F64, VFMS.F64, VFNMS.F64
3) one of the following subset of NEON instructions: VABA, VABAL, VPADAL, VRADDHN, VRSRA, VRSUBHN, VSWP, VTBL, VTBX, VTRN, VUZP, VZIP, VCVT.F32.F16, VCVT.F16.F32, VPADD, VPMIN, VPMAX, VABDL, VADDHN, VADDL, VADDW, VEXT, VMLAL, VMLSL, VMULL, VQDMLAL, VQDMLSL, VQDMULL, VSHLL, VSUBHN, VSUBL, VSUBW, that has a RAW or WAW hazard with 1.

In each case there can be other instructions between the initial floating point instruction and the NEON instruction. However none of the failing cases will occur if one of the intermediate instructions has a RAW or WAW register dependency on the initial floating point instruction.

The particular code sequences necessary to trigger this erratum are not anticipated and have not been observed in normal code to date. The presence of a data dependency between an FP operation and a NEON operation would be

highly unusual. However, it cannot be guaranteed that all current or future versions of compilers would not generate said sequences.

Implications

If any of cases 1, 2 or 3 of this erratum occur, the result of the NEON operation will be incorrect.

If case 4 of this erratum occurs the processor will deadlock.

Workaround

There is no workaround for this erratum.

2.5. Category B

745469: Data prefetcher can cause a deadlock when shared and non-shared data is resident in the data cache

Category B

Products Affected: Cortex-A5 MPCore.

Present in: r0p0

Description

Cortex-A5 MPCore includes an L1 data prefetcher which attempts to bring non-shareable data into the L1 data cache before it is required, to improve performance. It does not prefetch shareable memory.

Under rare circumstances, when both shareable and non-shareable data is being stored in the cache, the L1 data prefetcher may cause a deadlock.

Conditions

- The data cache is enabled.
- The SMP bit in the Auxiliary Control Register is set.
- The L1 data prefetcher is enabled (this is the default).
- A store is executed to a location of memory that is marked as cacheable and shareable.
- A sequence of load instructions is executed to memory that is marked as cacheable and not shareable. The load instructions miss in the cache, and are to addresses that cause the L1 data prefetcher to detect a pattern and start prefetching.

Implications

When the L1 data prefetcher is active the processor may occasionally deadlock.

Workaround

The workaround for this erratum is to disable the L1 data prefetcher. This can be done by clearing bits [14:13] of the Auxilliary Control Register. Disabling the L1 data prefetcher will reduce the performance of some code sequences when running from non-shareable memory, however it will not affect the performance of anything accessing shareable memory.

761273: Shareable cacheable write data may not automatically become visible to ACP

Category B

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

The Cortex-A5 MPCore Store Buffer has a mechanism to drain stores after a period of time to ensure that they become visible to the rest of the system. However, due to this erratum, in some limited circumstances this mechanism does not operate correctly and the store buffer does not drain. Consequently written data may remain in this buffer, invisible to the rest of the system outside of the processor.

If an external agent connected to the Accelerator Coherency Port (ACP) continually polls this memory location, waiting to see the update of the written data to make any further progress, then a system livelock may happen.

Conditions

- The Cortex-A5 processor is implemented with the optional ACP present.
- A CPU core in the processor does a store that is word, halfword or byte sized.
- The store is to an address that is in a Shareable Cacheable memory region.
- The CPU core in the processor does not execute a DMB, DSB, WFI, WFE, LDREX, LDREXD, LDREXH, LDREXB, SWP or SWPB instruction at any time after the store.
- An external agent connected to the Accelerator Coherency Port is polling the same address, waiting to observe the store.
- No other CPU core in the Cortex-A5 cluster accesses the cache line containing the address being polled.

Implications

Due to the erratum, a livelock situation may be encountered in the system.

Workaround

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

2.6. Category B (Rare)

743633: Memory requests from out-of-date page mappings may be observable after a forwarded TLB invalidate operation is executed

Category B Rare

Products Affected: Cortex-A5 MPCore.

Present in: r0p0

Description

A TLB invalidate operation is defined to be complete only when all memory accesses using the TLB entries that have been invalidated have been observed by all observers to the extent that those accesses are required to be observed. However if a TLB invalidate operation is executed on one CPU at the same time as a load or store instruction to the virtual address being invalidated is executed on another CPU, then due to this erratum the TLB operation may complete before all of the load or store instruction has completed.

Conditions

1) The Cortex-A5 MPCore system is configured with 2 or more CPUs

2) CPU A executes a load or store instruction that accesses more than one aligned word of memory. This could be an LDM, STM, VLDM, VSTM, LDRD, or STRD instruction that accesses multiple words of memory, or it could be an unaligned LDR, STR, LDRH, STRH, VLDR, VSTR, VLD, or VST instruction that crosses a word boundary.

3) CPU B writes to the page tables to modify the translation for the virtual address used by the instruction executing on CPU A. Note that due to TLB caching this write may happen before or during the instruction being executed on CPU A.

4) CPU B executes an inner shareable CP15 TLB maintenance instruction that invalidates the virtual address of the load or store being executed on CPU A. This could be a TLBIALLIS or TLBIASIDIS, a TLBIMVAAIS to the same address, or TLBIMVAIS to the same address and the same ASID.

5) The TLB invalidate instruction is executed at the same time as the load or store is being executed on CPU A.

6) CPU B executes a DSB instruction. There may be other instructions executed between the TLB invalidate and the DSB.

Implications

The implications of this erratum depend on whether the operating system makes full use of the architectural guarantees provided by the completion of a TLB invalidate operation. If it does, then this erratum could cause part of the load or store to happen using an old address translation after the operating system thinks that all accesses using that old translation have completed.

In the majority of the cases when this erratum occurs, especially when accessing cacheable memory, the load or store executing on CPU A will have completed naturally before the TLB invalidate and DSB complete, and so the effect of this erratum would not be visible to software.

Workaround

If the operating system does not rely on the completion guarantees provided by a TLB invalidate operation then no workaround is necessary. If the operating system does depend on the completion of the operation, then it must ensure that CPU A has finished executing the load or store by performing the following sequence:

1) Execute the TLB invalidate operation.

2) Send an interrupt request to CPU A. The interrupt handler on CPU A does not need to perform any work; it can return immediately, because as a side effect of taking the interrupt the pipeline will be flushed.

3) When CPU B has seen that the interrupt has been taken, it should execute a second TLB invalidate operation. This could be the same operation as the first time, or it could be a different operation type as long as it is still an inner shareable operation.

4) CPU B can then execute the DSB.

744929: Unexpected instruction prefetch abort can occur in privileged modes

Category B Rare

Products Affected: Cortex-A5 MPCore.

Present in: r0p0

Description

The processor will incorrectly take a prefetch abort on a mis-predicted or not-predicted branch if there is a mode changing CPS instruction in the decode stage of the pipeline that would take the processor in to User mode and the micro-TLB entry that is hit is marked as privileged mode only.

This erratum can only occur on a specific subset of branch instructions and will not occur on any kind of exception. The branches that are affected are any direct or indirect branch that is either mis-predicted or not predicted by the branch prediction unit. The mis-prediction could take the form of taken/not-taken speculation or target address mis-compare for the indirect branches that hit in the Call/Return Stack or Branch Target Address Cache. Unconditional branches cannot trigger this erratum.

This erratum requires a CPS instruction that changes to User mode or a literal pool in the branch shadow that is encoded as a CPS instruction that changes to User mode. The CPS instruction must lie in the shadow of the branch that is mis-predicted or not-predicted.

This erratum is not a security issue as it can only occur if the processor stays in the same security state throughout.

Conditions

Under the following conditions a prefetch abort will be taken when it should not:

1) The processor is in a secure privileged mode or non-secure kernel mode.
2) A branch is identified as mis-predicted or not predicted, resulting in a pipeline flush from the Writeback (Wr) or Retire (Ret) pipeline stages and instruction fetch from the corrected address.
3) In the same cycle as the flush, a CPS instruction to change the processor mode to User is in the Decode (De) stage of the pipeline.
4) On the cycle after the flush, and for one cycle only, the dpu_mode_iss bus that is signalled to the PreFetch Unit (PFU) from the Data-Processing Unit (DPU) will indicate User mode rather than a privileged mode.
5) In this cycle the PFU looks up the corrected address in the micro-TLB which hits.
6) The micro-TLB entry contains an entry that is marked as privileged mode only which triggers a prefetch abort.

Implications

The processor will take a Prefetch abort signalled as a permission fault when it should not.

Workaround

The effects of the erratum can be nullified when the software running on the processor includes support for handling prefetch aborts, such as that common in Operating Systems. The Prefetch abort handler can determine that the exception is extraneous by checking for a permission fault when the processor was not in User mode (indicated by SPSR_abt) and returning early from the exception to retry the instruction fetch.

753869: VFPv4 and NEON fused multiply accumulate result may be incorrectly signed when addend is infinite

Category B Rare

Products Affected: Cortex-A5 NEON Media Engine.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes support for fused multiply accumulate operations as part of the Floating-Point Unit and NEON Media engine.

Due to this erratum these instructions can produce incorrect results under certain conditions and operand values.

Conditions

Either:

1) The processor executes a single precision VFPv4 fused multiply accumulate or subtract instruction (VFMA.F32, VFMS.F32, VFNMA.F32 or VFNMS.F32)
2) Addend must be infinite and positive if in Round towards Minus Infinity (RM) mode (bits[22:23] of the FPSCR set to 0b10) or negative otherwise
3) For VFMA.F32 and VFNMA.F32 the product must have the opposite sign to the addend
4) For VFMS.F32 and VFNMS.F32 the product must have the same sign as the addend

Or:

1) The processor executes a NEON fused multiply accumulate or subtract instruction (VFMA.F32 and VFMS.F32)
2) Addend must be negative infinity
3) For VFMA.F32 the product must have the opposite sign to the addend
4) For VFMS.F32 the product must have the same sign as the addend

In both cases the instruction operands must meet the following criterion:

- The two multiplicands must have a product that is exactly ±2^128

Example:

```
VFMA.F32 S1, S2, S3; FPSCR.RM = 0b00
```

Before operation,

S1 = 0xFF800000 (= -Infinity)

S2 = 0x5F800000 (= 2^64)

S3 = 0x5F800000 (= 2^64)

In Round-to-Nearest rounding mode the result should be: 0xff800000 (-Infinity)

Due to this erratum the processor will produce the result: 0x7f800000 (+Infinity)

Implications

If the criteria described above are true then the result of the instruction will be incorrect. The result of the calculation will be infinity, as expected, but the sign will be incorrect.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

753870: VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded

Category B Rare

Products Affected: Cortex-A5 NEON Media Engine.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes double precision fused multiply accumulate operations as part of the VFPv4 support in the NEON Media Engine.

Due to this erratum these instructions can produce incorrect numerical results under certain conditions and operand values.

Conditions

For this erratum to occur one of the following cases must occur in Double Precision fused multiply accumulate instructions executed in either ARM or Thumb state:

Case 1

1) The Floating-Point Unit is configured in Round-to-Nearest (RN) rounding mode (bits[23:22] of the FPSCR are set to 0b00)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) The addend is an exact power of two (bits[51:0] of the operand are zero)
4) The product of the two multiplicands is of a magnitude such that the most significant bit (MSB) of the product is two places below the least significant bit (LSB) of the addend e.g.
   - Addend: 1.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   - Product: 0.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 011x
5) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend
6) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3; FPSCR.RM =0b00
```

Before operation,

D1 = 0x4330000000000000 (= 2^52 = 4503599627370496)

D2 = 0x3fe8000000000000 (= 0.75)

D3 = 0x3fd8000000000000 (= 0.375)

Result of operation D1 – (D2 * D3) to infinite precision is:

4503599627370496 – (0.75 * 0.375) = 4503599627370496 – 0.28125 = 4503599627370495.71875

In Round-to-Nearest rounding mode the result should be: 4503599627370495.5

Due to this erratum, the processor will produce the result: 4503599627370496

Case 2

1) The Floating-Point Unit is not configured in Flush-to-zero mode (bit[24] of the FPSCR is not set)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) One multiplicand is the minimum denormal number
4) The other multiplicand must have a large enough exponent that the final product is representable as a normal number
5) The exponent of the addend must be 1126 smaller than the larger multiplicand
6) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend

7) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3
```

Before operation,

D1 = 0x29ac000000000000

D2 = 0x7000000000000000

D3 = 0x0000000000000001

FPSCR = 0x00000000

Expected result:

D1 = 0xacdfffffffffffc

FPSCR = 0x00000010

Result on Cortex-A5:

D1 = 0xacdfffffffffffd

FPSCR = 0x00000010

Case 3

1) The Floating-Point Unit is configured in Flush-to-zero mode (bit[24] of the FPSCR is set)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) The absolute value of the addend is less than the absolute value of the product of the two multiplicands
4) The result of the operation before rounding should be subnormal and such that if Flush-to-zero were not enabled, the result would be rounded to the minimum normal number
5) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend
6) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3; FPSCR.FZ = 1
```

Before operation,

D1 =0x0000000000000000

D2 =0x001fffffffffffff

D3 =0x3fe0000000000000

FPSCR = 0x01000000

Expected result:

D1 = 0x8000000000000000

FPSCR = 0x01000008

Result on Cortex-A5:

D1 = 0x8010000000000000

FPSCR = 0x01000010

Implications

If the criteria described in any of the cases above are true then the result of the instruction will be numerically incorrect.

In cases 1 and 2 the value obtained from the calculation will be incorrectly rounded, giving a result 1 unit of least precision (ULP) greater than expected. The inexact exception flag (bit[4] of FPSCR) will be correctly set to 1.

In case 3 the value obtained from the calculation will not be flushed to zero and instead will be rounded to the minimum normal number. The underflow exception flag (UFC, bit[3] of FPSCR) and the inexact flag (IXC, bit[4] of the FPSCR) will also be incorrect. The correct result should be that UFC should be set and the IXC should be unaffected. Due to this erratum the UFC will be unchanged and the IXC will be set.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

761269: Exclusive monitor might incorrectly remain set in one CPU core when another has exclusive access

Category B Rare

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

The Cortex-A5 MPCore processor implements an exclusive monitor to support the execution of load and store exclusive instructions. This monitor tracks access to a cache line to determine whether any other CPU core has written to the same address during the load/store exclusive sequence. Under certain rare conditions described below, the monitor might incorrectly remain set when another CPU core could be writing to the same address.

Configurations Affected

To be affected by this erratum, the following must apply:

1) The processor is implemented with two or more CPU cores

Conditions

- A linefill of shareable cacheable memory is started on CPU core 0 by a non-exclusive load, or by a speculative data access
- A LDREX, LDREXD, LDREXH, or LDREXB instruction is executed on CPU core 0, at the same time as the linefill is in progress. The address used by the load exclusive instruction must be in the same cache line as the linefill.
- The cache line accessed by the exclusive load is evicted at the same time as the load exclusive is executing with specific micro-architectural conditions and timing or the cache line is not allocated because an external abort was reported on the AXI interface.
- Another CPU core in the processor performs a store to the same address as the load exclusive
- The line is allocated back into the cache of CPU core 0
- CPU core 0 executes a STREX, STREXD, STREXH or STREXB instruction. This instruction might incorrectly pass the exclusive monitor.

The erratum will only have an effect if the line is read back into the data cache of CPU core 0 after the load exclusive has completed, but before the store exclusive has started. This can only occur if:

- An explicit load or store instruction is executed after the load exclusive instruction and before the corresponding store exclusive instruction. This load or store instruction can be conditional but it must pass its condition codes
- A linefill is started by a TLB pagewalk. This requires that the processor is programmed to perform cacheable pagewalks, and the load/store exclusive sequence is being performed to an address that is part of the translation table

Implications

If the erratum occurs, then a store exclusive instruction might succeed when it should have failed. If software is using the exclusive sequence to implement a synchronization mechanism such as a semaphore, then it could result in two CPU cores in the processor both obtaining the semaphore at the same time.

However the ARM architecture requires that correctly written software must not include any instructions between the load exclusive and store exclusive instructions which could result in explicit memory accesses. If this requirement is met then this erratum can only occur if the virtual address used by the exclusive instructions is in shareable cacheable memory and maps to a physical address which is located in an active translation table, and the linefill is started by a TLB walk.

Note: This erratum will not effect a code sequence used as part of a compare-and-swap/compare-and-exchange primitive where a load or store is conditionally executed after the load exclusive in a path where the store exclusive is not explicitly executed; for example:

```
MOV    r1,#<new_val>
LDREX r0,[A1]
CMP    r0,#<old_val>
```

```
        BNE    changed
        STREX r0,r1,[A1]
        BX     lr
changed LDR    r2,[A1]      ; A load in this branch cannot cause this erratum
        BX     lr
```

Workaround

If the software meets the requirements of the architecture for load exclusive and store exclusive instructions and the address used is not part of the translation tables then no workaround is required for this erratum. It is expected that this will be the case for the majority of software.

If exclusive operations are required to be performed to addresses that are part of the active translation tables which map shareable cacheable memory then the erratum can be avoided by setting the translation table to inner non-cacheable in the TTBR register. The translation tables can still be marked as outer cacheable.

Alternatively a separate address can be be used to hold the semaphore when atomic access to translation table entries is required.

765872: VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded

Category B Rare

Products Affected: Cortex-A5 Floating-Point Unit.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes support for double precision fused multiply accumulate operations as part of the Floating-Point Unit.

Due to this erratum these instructions can produce incorrect numerical results under certain conditions and operand values.

Conditions

For this erratum to occur one of the following cases must occur in Double Precision fused multiply accumulate instructions executed in either ARM or Thumb state:

Case 1

1) The Floating-Point Unit is configured in Round-to-Nearest (RN) rounding mode (bits[23:22] of the FPSCR are set to 0b00)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) The addend is an exact power of two (bits[51:0] of the operand are zero)
4) The product of the two multiplicands is of a magnitude such that the most significant bit (MSB) of the product is two places below the least significant bit (LSB) of the addend e.g.
   - Addend: 1.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   - Product: 0.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 011x
5) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend
6) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3; FPSCR.RM =0b00
```

Before operation,

D1 = 0x4330000000000000 (= 2^52 = 4503599627370496)

D2 = 0x3fe8000000000000 (= 0.75)

D3 = 0x3fd8000000000000 (= 0.375)

Result of operation D1 – (D2 * D3) to infinite precision is:

4503599627370496 – (0.75 * 0.375) = 4503599627370496 – 0.28125 = 4503599627370495.71875

In Round-to-Nearest rounding mode the result should be: 4503599627370495.5

Due to this erratum, the processor will produce the result: 4503599627370496

Case 2

1) The Floating-Point Unit is not configured in Flush-to-zero mode (bit[24] of the FPSCR is not set)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) One multiplicand is the minimum denormal number
4) The other multiplicand must have a large enough exponent that the final product is representable as a normal number
5) The exponent of the addend must be 1126 smaller than the larger multiplicand
6) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend

7) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3
```

Before operation,

D1 = 0x29ac000000000000

D2 = 0x7000000000000000

D3 = 0x0000000000000001

FPSCR = 0x00000000

Expected result:

D1 = 0xacdfffffffffffc

FPSCR = 0x00000010

Result on Cortex-A5:

D1 = 0xacdfffffffffffd

FPSCR = 0x00000010

Case 3

1) The Floating-Point Unit is configured in Flush-to-zero mode (bit[24] of the FPSCR is set)
2) The processor executes a double precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F64, VFMS.F64, VFNMA.F64 or VFNMS.F64)
3) The absolute value of the addend is less than the absolute value of the product of the two multiplicands
4) The result of the operation before rounding should be subnormal and such that if Flush-to-zero were not enabled, the result would be rounded to the minimum normal number
5) For a VFMA.F64 or VFNMA.F64, the product is of opposite sign to the addend
6) For a VFMS.F64 or VFNMS.F64, the product is of the same sign as the addend

Example:

```
VFMS.F64 D1, D2, D3; FPSCR.FZ = 1
```

Before operation,

D1 =0x0000000000000000

D2 =0x001fffffffffffff

D3 =0x3fe0000000000000

FPSCR = 0x01000000

Expected result:

D1 = 0x8000000000000000

FPSCR = 0x01000008

Result on Cortex-A5:

D1 = 0x8010000000000000

FPSCR = 0x01000010

Implications

If the criteria described in any of the cases above are true then the result of the instruction will be numerically incorrect.

In cases 1 and 2 the value obtained from the calculation will be incorrectly rounded, giving a result 1 unit of least precision (ULP) greater than expected. The inexact exception flag (bit[4] of FPSCR) will be correctly set to 1.

In case 3 the value obtained from the calculation will not be flushed to zero and instead will be rounded to the minimum normal number. The underflow exception flag (UFC, bit[3] of FPSCR) and the inexact flag (IXC, bit[4] of the FPSCR) will also be incorrect. The correct result should be that UFC should be set and the IXC should be unaffected. Due to this erratum the UFC will be unchanged and the IXC will be set.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

765873: VFPv4 fused multiply accumulate result may be incorrectly signed when addend is infinite

Category B Rare

Products Affected: Cortex-A5 Floating-Point Unit.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes support for fused multiply accumulate operations as part of the Floating-Point Unit.

Due to this erratum these instructions can produce incorrect results under certain conditions and operand values.

Conditions

1) The processor executes a single precision VFPv4 fused multiply accumulate or subtract instruction (VFMA.F32, VFMS.F32, VFNMA.F32 or VFNMS.F32)
2) Addend must be infinite and positive if in Round towards Minus Infinity (RM) mode (bits[22:23] of the FPSCR set to 0b10) or negative otherwise
3) For VFMA.F32 and VFNMA.F32 the product must have the opposite sign to the addend
4) For VFMS.F32 and VFNMS.F32 the product must have the same sign as the addend
5) The two multiplicands must have a product that is exactly ±2^128

Example:

```
VFMA.F32 S1, S2, S3; FPSCR.RM = 0b00
```

Before operation,

S1 = 0xFF800000 (= -Infinity)

S2 = 0x5F800000 (= 2^64)

S3 = 0x5F800000 (= 2^64)

In Round-to-Nearest rounding mode the result should be: 0xff800000 (-Infinity)

Due to this erratum the processor will produce the result: 0x7f800000 (+Infinity)

Implications

If the criteria described above are true then the result of the instruction will be incorrect. The result of the calculation will be infinity, as expected, but the sign will be incorrect.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

769221: Stores to out-of-date translation table mappings might be observable after a TLB invalidate operation completes

Category B Rare

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

A TLB invalidate operation is defined to be complete only when all memory accesses using the TLB entries that have been invalidated have been observed by all observers to the extent that those accesses are required to be observed. However if a TLB invalidate operation is executed on one CPU core at the same time as a store to the virtual address being invalidated is in progress on another CPU core, then this erratum means the TLB operation might complete before the store completes.

Configurations Affected

To be affected by this erratum, the following must apply:

- The processor is implemented with two or more CPU cores.

Conditions

1) CPU core 0 executes a store operation to virtual address VA1, that is marked as shared.
2) CPU core 1 writes to the translation tables to modify the translation that maps virtual address VA1 to physical address PA1.
3) CPU core 1 performs an inner shareable CP15 TLB maintenance operation that invalidates the mapping of VA1. This could be TLBIALLIS, TLBIASIDIS, TLBIMVAAIS to the same virtual address, or TLBIMVAIS to the same virtual address and the same ASID.
4) CPU core 1 executes a DSB instruction. It might execute other instructions between the TLB invalidate and the DSB.
5) CPU core 1 alters memory at the physical address PA1 that was previously accessible from virtual address VA1.
6) The store in CPU 0 both:

- Takes longer to complete than all outstanding data loads and stores that are in progress at the time the TLB invalidate operation is received by CPU core 0
- Does not complete until after the memory at address PA1 has been altered.

Notes:

- There are very specific timing requirements for this erratum to occur; the store must be in a specific state in CPU core 0 on the cycle at which the TLB operation is received.
- This erratum cannot affect a store executed on the same CPU core as the TLB invalidate operation.

Implications

The implications of this erratum depend on whether the operating system makes full use of the architectural guarantees of the completion of a TLB invalidate operation.

If the store is to cacheable memory and the physical address PA1 is present in the data cache of CPU core 0 then this erratum will have no effect. However if the store is to non-cacheable memory, or if the physical address is not present in the data cache of CPU core 0 then CPU core 0 might either:

- Write to PA1, possibly overwriting the contents of the external memory
- Read new (unrelated) data, for VA1, from the new mapping set up by CPU core 1

Workaround

If the operating system does not rely on the completion guarantees across multiple observers provided by a TLB invalidate operation then no workaround is necessary. If the operating system does depend on the completion of the operation, then it must ensure that the store on CPU core 0 has completed before CPU core 1 performs the translation table change. It ensures the store by performing the following sequence on CPU core 1:

1) Execute the (shared) TLB invalidate; DSB sequence as given in Conditions 3 and 4.

2) Execute the following sequence of instructions:

- TLBIMVAIS
- DSB.

The TLBIMVAIS can use any virtual address.

If the TLB maintenance procedure in the software includes a number of invalidate operations, for example over a range of virtual addresses/ASID in a loop, then the workaround only needs to be applied once at the end of the loop.

2.7. Category C

745769: Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0

Description

On Cortex-A5 some floating point instructions which take a large number of cycles to calculate results are allowed to complete out-of-order with the executed instruction stream as long as there are no outstanding dependencies. If such a dependency is reached the processor pipeline must stall until the result of the calculation is available.

This erratum can occur when there is an outstanding floating point divide or square-root operation present when the processor takes an Array-bounds or Null-pointer exception in Jazelle DBX state. The Jazelle DBX exception must occur within 30 cycles from the point at which the floating point instruction is committed as this is the maximum number of cycles one of these operations can take to complete. Under these circumstances, if the completion of the floating point instruction and the Jazelle DBX exception overlap the link register will not be updated.

The result of the floating point divide or square-root operation is always written correctly. This erratum cannot occur on a Cortex-A5 processor implemented without Jazelle DBX.

Conditions

1) A VSQRT/VDIV is committed to being executed
2) Jazelle DBX state is entered within 30 cycles and without the result of the VSQRT/VDIV having been used
3) A Jazelle DBX instruction is executed that initiates a Null-pointer or Array-bounds Jazelle DBX exception
4) The VSQRT / VDIV that had not finished executing when Jazelle DBX state was entered writes its result
5) The Null-pointer or Array-bounds Jazelle DBX exception completes.

Implications

If this erratum occurs the return address for the exception will not be correct in the Jazelle exception handler, resulting in incorrect execution of software following the return from the exception.

Workaround

The erratum can be avoided by executing a `VMSR FPSID, Rt` instruction before entering Jazelle DBX state. This will guarantee that the result of the VSQRT / VDIV instruction is written before Jazelle DBX state is entered.

753670: VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode

Category C

Products Affected: Cortex-A5 NEON Media Engine.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes support for fused multiply accumulate operations as part of the Floating-Point Unit and NEON Media engine.

Due to this erratum these instructions can produce incorrect results under certain conditions and operand values.

Conditions

Either:

1) The Floating-Point Unit is configured in Flush-to-zero mode (bit[24] of the FPSCR is set)

2) The processor executes a single precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F32, VFMS.F32, VFNMA.F32 or VFNMS.F32)

Or:

1) The processor executes a NEON fused multiply accumulate or subtract instruction (VFMA.F32 and VFMS.F32)

In both cases the instruction operands must meet the following criteria:

- The addend must be non-zero and de-normal
- The multiplicands must be non-zero and normal
- The operand values are such that if the equivalent operation was performed in VFPv4 and not in Flush-to-zero mode (bit[24] of the FPSCR is clear) the results would be zero and exact. That is, the accumulation is performed as Unlike-Signed-Addition and complete cancellation occurs.

These conditions are unlikely to occur in practice as the addend will have normally been generated by a previous VFPv4 or NEON floating-point instruction. In either case the addend value will be flushed to zero either by default in the case of NEON instructions or by the action of Flush-to-zero mode.

Implications

If the criteria described above are true then the result of the instruction and the flags set in FPSCR will be incorrect. The result will be non-zero and de-normal. The IDC flag (bit[7] of the FPSCR) will be correctly set indicating that the addend is de-normal. The UFC flag (bit[3] of the FPSCR) will not be updated correctly; UFC should be set to 1 as an underflow occurs in the intended calculation but due to this erratum it will not be changed in the FPSCR.

Note: If the result of the instruction is used as in a subsequent VFPv4 or NEON floating-point arithmetic or conversion instruction then the value will be flushed to zero, so any following calculations will yield the correct results.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

765870: A load from normal memory marked with a debug watchpoint may result in an external abort

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

Cortex-A5 implements synchronous watchpoints. According to the architecture a memory access that generates a synchronous watchpoint debug event must not generate an external abort.

Due to this erratum the processor can take an external abort on a read to normal cacheable or non-cacheable memory marked with a watchpoint if the abort information was previously stored in a line-fill buffer.

Conditions

A watchpoint must be set on a virtual address mapped in the MMU to a physical address which will result in either an AXI Slave or Decode error when presented to the external memory system.

If the memory is mapped as normal non-cacheable the following conditions are required for the erratum to occur:

1) The processor must execute a load multiple instruction with at least two registers in the list or a load double instruction
2) A watchpoint hit is generated for any access in the load apart from the first access
3) The watchpoint hit is generated for an access with address VA where VA[4:0] != 0.

If the memory is mapped as normal cacheable the following conditions are required for the erratum to occur:

1) The processor must execute a load instruction of any type
2) The data associated with the load must not be present in the level 1 data cache
3) The data associated with the load must have been previously fetched from external memory by an unrelated access, for example:
    1. A previous load to the same cache line, either single or multiple, or
    2. A data pre-fetch when this functionality is not disabled (ACTLR[14:13] != 0b00).

If the criteria described above are true then the instruction will take an external abort exception rather than generating a watchpoint debug event.

The erratum will not occur in memory marked as Strongly-Ordered or Device.

Implications

Due to this erratum it will not be possible to guarantee that a watchpoint event will occur when the associated address is in normal memory and will generate an external abort. This means that a debugger cannot trap such an address using the watchpoint mechanism.

Workaround

A debugger can trap a data abort using the vector-catch feature of the debug architecture. If one of these exceptions occurs when watchpoints are active the software can read the aborting address and type from the DFAR and DFSR system registers and determine whether it was caused by an external abort and also matches one of the current watchpointed addresses.

765871: Watchpointed access in a store-multiple is not masked

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

Cortex-A5 implements synchronous watchpoints. A synchronous watchpoint generated during a store multiple instruction must ensure that the watchpointed accesses do not perform writes on the bus to update memory. Due to this erratum this requirement is not met and the processor will incorrectly update memory for a watchpointed access.

Conditions

All the following conditions are required for this erratum to occur:

1) A store multiple instruction is executed with at least 2 registers to be stored
2) The store multiple instruction writes to memory defined as Strongly-Ordered or Device
3) A watchpoint hit is generated for any access in the store multiple apart from the first access
4) The watchpoint hit is generated for an access with address bits[4:0] != 0x0

In these cases the store multiple will continue to perform writes until either the end of the store multiple or the end of the current cache line.

Implications

Due to this erratum, the memory contents of the watchpointed address are updated before the debug event can be recognised. This means that a debugger:

1) Cannot always assume memory has not been updated when a watchpoint generates a debug event
2) Cannot prevent an access by setting a watchpoint on it

The ARM architecture recommends that watchpoints should not be set on individual device or strongly ordered addresses that can be accessed as part of a load or store multiple. Instead, it recommends the use of the address range masking functionality provided to set watchpoints on an entire region, ensuring that the watchpoint event will be seen on the first access of a load or store multiple to this region.

If this recommendation is followed, this erratum will not occur.

Workaround

There is no workaround for this erratum.

765874: VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode

Category C

Products Affected: Cortex-A5 Floating-Point Unit.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes support for fused multiply accumulate operations as part of the Floating-Point Unit.

Due to this erratum these instructions can produce incorrect results under certain conditions and operand values.

Conditions

1) The Floating-Point Unit is configured in Flush-to-zero mode (bit[24] of the FPSCR is set)
2) The processor executes a single precision VFPv4 fused-multiply accumulate or subtract instruction (VFMA.F32, VFMS.F32, VFNMA.F32 or VFNMS.F32)
3) The addend must be non-zero and de-normal
4) The multiplicands must be non-zero and normal
5) The operand values are such that if the equivalent operation was performed in VFPv4 and not in Flush-to-zero mode (bit[24] of the FPSCR is clear) the results would be zero and exact. That is, the accumulation is performed as Unlike-Signed-Addition and complete cancellation occurs.

These conditions are unlikely to occur in practice as the addend will have normally been generated by a previous VFPv4 floating-point instruction, in which case the addend value will be flushed to zero by the action of Flush-to-zero mode.

Implications

If the criteria described above are true then the result of the instruction and the flags set in FPSCR will be incorrect. The result will be non-zero and de-normal. The IDC flag (bit[7] of the FPSCR) will be correctly set indicating that the addend is de-normal. The UFC flag (bit[3] of the FPSCR) will not be updated correctly; UFC should be set to 1 as an underflow occurs in the intended calculation but due to this erratum it will not be changed in the FPSCR.

Note: If the result of the instruction is used in a subsequent VFPv4 floating-point arithmetic or conversion instruction then the value will be flushed to zero, so any following calculations will yield the correct results.

Workaround

There is no general workaround for this erratum. However:

- If the additional precision of fused-MAC is not required the instructions can be replaced by the equivalent chained-MAC operations i.e. replace VFMA with VMLA, VFMS with VMLS etc
- If performance is not critical the erratum conditions can be trapped by examining the input operand values as the failing conditions are always deterministic

767976: Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

A TLB invalidate operation is defined to be complete only when all memory accesses using the TLB entries that have been invalidated have been observed by all observers to the extent that those accesses are required to be observed. However if a TLB invalidate operation is executed at the same time as an instruction fetch from the virtual address being invalidated is in progress, on the same CPU core or on another CPU core, then this erratum means the TLB operation might complete before the instruction fetch has completed

Configurations Affected

This erratum affects all configurations of the processor

Conditions

1) CPU core 0 is executing code from address VA1
2) CPU core 1 writes to the translation tables to modify the translation that maps virtual address VA1 to physical address PA1.
3) CPU core 1 executes an inner shareable CP15 TLB maintenance instruction that invalidates the mapping of VA1. This could be a TLBIALLIS, TLBIASIDIS, TLBIMVAAIS to the same address, or TLBIMVAIS to the same address and the same ASID.
4) CPU core 1 executes a DSB instruction. It might execute other instructions between the TLB invalidate and the DSB.
5) CPU core 1 alters memory at the physical address PA1 that was previously accessible from virtual address VA1.
6) The instruction fetch on CPU core 0 both:

- Takes longer to complete than all outstanding data loads and stores that are in progress at the time the TLB invalidate operation is received on CPU core 0
- Does not complete until after the memory at address PA1 has been altered and therefore reads the altered value.

Implications

The implications of this erratum depend on whether the operating system makes full use of the architectural guarantees of the completion of a TLB invalidate operation. If it does, then this erratum could cause an instruction fetch to happen using an old address translation after the operating system expects all accesses to be using the new address translation.

However it is not expected that software alters the translation table mappings of code regions without invalidating any references to the original mapping from the instruction cache. Because the cache is physically addressed, an instruction fetch from a new virtual address mapped to the

old physical address can result in a cache hit on stale data. If the software already does this invalidation as part of the remapping procedure, before any CPU core updates the physical memory at PA1, then there are no implications to this erratum.

Workaround

No workaround is necessary if either:

- The operating system does not rely on the completion guarantees provided by the TLB invalidate
- The operating system does depend on the TLB invalidate operation, but already includes an ISB and instruction cache invalidate as part of the routine to map the old address out before the memory is updated

Otherwise, the erratum can be worked around by performing the following sequence:

1) Execute the TLB invalidate operation and DSB
2) Execute an ISB instruction

3) Execute a cache invalidate operation ICIMVAU to a virtual address with a valid mapping on all CPU cores to memory marked as shared

4) Execute a DSB instruction to ensure that the instruction cache invalidate operation is complete.

781769: ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

The Cortex-A5 includes a number of architecturally defined CPUID identification registers that can be used by software to determine the supported features of the processor. This erratum means that, if the processor is implemented with only the trivial Jazelle implementation the value of the state2 field of the Processor Feature Register 0 (ID_PFR0) is incorrect. ID_PFR0[11:8] is 0x2 when it should be 0x1.

Note: The trivial Jazelle implementation means the processor does not include the hardware Java bytecode acceleration extension.

Configurations Affected

To be affected by this erratum, the processor must be implemented with only the trivial Jazelle implementation.

Implications

The ID_PFR0[11:8] value can be used in software in two ways; to check whether the processor includes the full Jazelle extension, and to determine if the CV bit in the Jazelle JOSCR register cleared on exception entry.

This erratum has no implications for the first use of the ID_PFR0[11:8] field.

Although the value in the field is incorrect it is non-zero and so correctly indicates the Jazelle extension is implemented.

There are also no implications of the second use of the ID_PFR0[11:8] field. The value can be used by the exception handler software to determine whether the JOSCR.CV needs to be cleared by software before re-entry into Java code, or whether this is done automatically in hardware. In the trivial implementation of Jazelle, clearing or not clearing JOSCR.CV has no effect because the processor does not support the execution of Java bytecodes in hardware.

Workaround

No workaround is required for this erratum as the erratum has no implications.

791671: A Non-Cacheable store in a tight loop might not become observable until the loop completes

Category C

Products Affected: Cortex-A5 MPCore.

Present in: r0p0, r0p1

Description

The ARM architecture states that all writes complete in a finite period of time in implementations that include the Multiprocessing Extensions. This applies for all writes, including repeated writes to the same region. Due to this erratum, an uninterrupted pattern of stores to the same Non-Cacheable 32-byte aligned memory region might not become observable to another CPU or master in the system.

Configurations Affected

This erratum affects all configurations of the Cortex-A5 MP Core processor.

Conditions

A CPU Core in the processor executes a loop containing only one or more stores to the same Normal Non-Cacheable 32-byte aligned memory region.

If these conditions are met, then the stores might continuously merge inside the processor until the loop completes.

This erratum cannot occur if any instructions apart from the stores and the branch are present in the loop.

Implications

Another CPU Core in the processor or another master in the system that reads from the 32-byte aligned memory region might not receive the most up-to-date data until the loop completes

The erratum is not expected to be observed in real code for the following reasons:

- System timers and interrupts will normally change the program flow on the CPU Core executing the loop. This will cause the stores to become observable.
- The last store will become observable to other CPU Cores and masters in the system when the loop completes.
- Polled variables that are being updated in a loop are likely to contain a barrier or a power-saving measure such as WFE or WFI.
- Loops will normally contain instructions between stores to the same 32-byte aligned memory region.

Workaround

A workaround is not expected to be necessary in real code. However, if a workaround is required then an instruction can be inserted into the loop to avoid the erratum.

If the software containing the loop cannot be modified then the recommended workaround is to force the CPU Core to regularly take an interrupt which would act as a watchdog. Several options are possible to generate this regular interrupt, which might be specific to each system.

Possible candidates are interrupts generated by a local timer, global timer, or PMU cycle counter overflow.