

ARM Processor CortexTM-A5

Product Revision r0

Software Developers Errata Notice

Non-Confidential - Released



Software Developers Errata Notice

Copyright © 2012-2014 ARM. All rights reserved.

Non-Confidential Proprietary Notice

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

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

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

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

Copyright © 2012-2014 ARM Limited 110 Fulbourn Road, Cambridge, England CB1 9NJ. All rights reserved.

Web Address

<http://www.arm.com>

Feedback on content

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

- the document title
- the document number, ARM-EPM-025157
- 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.

29 May 2014: Changes in Document v3

Page	Status	ID	Cat	Rare	Summary of Erratum
33	New	807269	CatC		PMU event counters 0x07, 0x0C, 0x0E and 0x14 do not increment correctly

29 Apr 2013: Changes in Document v2

Page	Status	ID	Cat	Rare	Summary of Erratum
31	New	803371	CatC		Reads of the DBGPCSR can cause incorrect values to be latched into DBGCIDSR
32	New	803570	CatC		TBB/TBH incorrect branch address is not mis-predicted

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	744574	CatB	Rare	Unexpected instruction prefetch abort can occur in privileged modes
11	New	753869	CatB	Rare	VFPv4 and NEON fused multiply accumulate result may be incorrectly signed when addend is infinite
12	New	753870	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded
15	New	765872	CatB	Rare	VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded
18	New	765873	CatB	Rare	VFPv4 fused multiply accumulate result may be incorrectly signed when addend is infinite
19	New	730105	CatC		Watchpoint not taken on VM Implementation Table entry for Array Bounds and Null Pointer Jazelle Exceptions
20	New	732433	CatC		Mismatched pagetable aliases can cause a deadlock
21	New	738097	CatC		Conditional VMRS APSR_nzcv, FPSCR may evaluate with incorrect flags
22	New	745374	CatC		Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding
23	New	753670	CatC		VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode
24	New	756269	CatC		Watchpointed access in a store-multiple is not masked
25	New	756274	CatC		A load from normal memory marked with a debug watchpoint may result in an external abort
26	New	765874	CatC		VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode
27	New	769220	CatC		ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation
28	New	769870	CatC		Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes
30	New	791672	CatC		A Non-Cacheable store in a tight loop might not become observable until the loop completes

Contents

CHAPTER 1.	5
INTRODUCTION	5
1.1. Scope of this document	5
1.2. Categorization of errata	5
CHAPTER 2.	6
ERRATA DESCRIPTIONS	6
2.1. Product Revision Status	6
2.2. Revisions Affected	6
2.3. Category A	8
2.4. Category A (Rare)	8
741950: Data hazards between non-NEON FP and certain NEON instructions can deadlock the processor under certain conditions.....	8
2.5. Category B	9
2.6. Category B (Rare)	9
744574: Unexpected instruction prefetch abort can occur in privileged modes.....	9
753869: VFPv4 and NEON fused multiply accumulate result may be incorrectly signed when addend is infinite	11
753870: VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded	12
765872: VFPv4 Double Precision fused multiply accumulate result may be incorrectly rounded	15
765873: VFPv4 fused multiply accumulate result may be incorrectly signed when addend is infinite	18
2.7. Category C	19
730105: Watchpoint not taken on VM Implementation Table entry for Array Bounds and Null Pointer Jazelle Exceptions.....	19
732433: Mismatched pagetable aliases can cause a deadlock.....	20
738097: Conditional VMRS APSR_nzcv, FPSCR may evaluate with incorrect flags	21
745374: Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding.....	22
753670: VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	23
756269: Watchpointed access in a store-multiple is not masked.....	24
756274: A load from normal memory marked with a debug watchpoint may result in an external abort	25
765874: VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	26
769220: ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation.....	27
769870: Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes	28
791672: A Non-Cacheable store in a tight loop might not become observable until the loop completes	30
803371: Reads of the DBGPCSR can cause incorrect values to be latched into DBGCIDSR.....	31
803570: TBB/TBH incorrect branch address is not mis-predicted	32
807269: PMU event counters 0x07, 0x0C, 0x0E and 0x14 do not increment correctly	33

Chapter 1.

Introduction

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

1.1. Scope of this document

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

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

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

1.2. Categorization of errata

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

Table 1 **Categorization of errata**

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

Chapter 2.

Errata Descriptions

2.1. Product Revision Status

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

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

2.2. Revisions Affected

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

This document includes errata that affect revision 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	
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
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
744574	CatB	Rare	Unexpected instruction prefetch abort can occur in privileged modes	X	
807269	CatC		PMU event counters 0x07, 0x0C, 0x0E and 0x14 do not increment correctly	X	X
803570	CatC		TBB/TBH incorrect branch address is not mis-predicted	X	X
803371	CatC		Reads of the DBGPCSR can cause incorrect values to be latched into DBGCIDSR	X	X
791672	CatC		A Non-Cacheable store in a tight loop might not become observable until the loop completes	X	X
769870	CatC		Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes	X	X
769220	CatC		ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation	X	X
765874	CatC		VFPv4 fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	X	X
756274	CatC		A load from normal memory marked with a debug watchpoint may result in an external abort	X	X
756269	CatC		Watchpointed access in a store-multiple is not masked	X	X

ID	Cat	Rare	Summary of Erratum	r0p0	r0p1
753670	CatC		VFPv4 and NEON fused multiply accumulate with de-normal operands may produce incorrect result in Flush-to-zero mode	X	X
745374	CatC		Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding	X	
738097	CatC		Conditional VMRS APSR_nzcv, FPSCR may evaluate with incorrect flags	X	
732433	CatC		Mismatched pagetable aliases can cause a deadlock	X	
730105	CatC		Watchpoint not taken on VM Implementation Table entry for Array Bounds and Null Pointer Jazelle Exceptions	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

There are no errata in this category

2.6. Category B (Rare)

744574: Unexpected instruction prefetch abort can occur in privileged modes

Category B Rare

Products Affected: Cortex-A5 Uni-Processor.

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 $\pm 2^{128}$

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
 - Product: 0.0000 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 = 0xacdffffffffffffc

FPSCR = 0x00000010

Result on Cortex-A5:

D1 = 0xacdffffffffffffd

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

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
 - 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 = 0xacdffffffffffffc

FPSCR = 0x00000010

Result on Cortex-A5:

D1 = 0xacdffffffffffffd

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 $\pm 2^{128}$

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

2.7. Category C

730105: Watchpoint not taken on VM Implementation Table entry for Array Bounds and Null Pointer Jazelle Exceptions

Category C

Products Affected: Cortex-A5 Uni-Processor.

Present in: r0p0

Description

CortexA5 will fail to generate a debug event when a watchpoint is programmed on loads of the VM Implementation Table entries for Array Bounds or Null Pointer Jazelle exceptions.

Conditions

- 1) Debug enabled; Monitor or Halting debug-mode
- 2) Watchpoint programmed on loads of the VM Implementation Table entries for Array Bounds / Null Pointer Jazelle Exception
- 3) Array Bounds / Null Pointer Jazelle Exception occurs

Implications

It is not possible to generate a debug event for the implicit load involved in taking an Array Bounds / Null Pointer Jazelle Exception.

Workaround

A breakpoint on the first instruction of the Array Bounds / Null Pointer Jazelle Exception handler will generate a debug event as expected.

732433: Mismatched pagetable aliases can cause a deadlock**Category C****Products Affected: Cortex-A5 Uni-Processor.****Present in: r0p0****Description**

If the pagetables are constructed such that two or more different virtual addresses map to the same physical address, but the virtual addresses have different memory types, then under certain conditions the processor may deadlock.

Conditions

- 1) A store is executed to normal non-cacheable, normal cacheable or device memory
- 2) A second store is executed to normal non-cacheable, normal cacheable or device memory
- 3) The physical address of the second store must be within the same 32 byte aligned address (i.e. the upper 27 bits of the address are the same, and the security state matches)
- 4) The memory type or cacheability attributes of the second store must be different to the first store
- 5) If either store is cacheable, then at least one of them must not hit in the cache
- 6) Other instructions may execute between the two stores, provided that there are no DMB or DSB instructions, and no accesses to strongly ordered memory
- 7) A load instruction to normal memory is executed
- 8) The physical address of the load must be within the same 32 byte aligned address as the store instructions
- 9) There must be no further loads, stores, DMB or DSB instructions between the second store and the load
- 10) The second store and the load must not be separated by more than one non-branch instruction

Implications

A deadlock can occur during execution of software which explicitly sets up two or more aliases to the same physical memory page with different inner or outer attributes.

It is not expected that many applications make use of mismatched memory types, and in most cases such use is architecturally UNPREDICTABLE.

One notable exception is the use case where the inner and outer attributes of the two aliases are identical apart from the outer cache-allocation fields. In Cortex-A5 one alias could be mapped to outer Normal, Write-Back Write-Allocate and the other to outer Normal, Write-Back, No Write-Allocate. If the inner cache attributes for both aliases are identical this mapping is architecturally correct, but can still provoke the errata conditions.

Workaround

If multiple aliases to the same physical page in memory are required, place a memory barrier instruction between load or store instructions to the aliased locations.

738097: Conditional VMRS APSR_nzcv, FPSCR may evaluate with incorrect flags**Category C****Products Affected: Cortex-A5 Uni-Processor.****Present in: r0p0****Description**

Under certain circumstances, a conditional VMRS APSR_nzcv, FPSCR instruction may evaluate its condition codes using the wrong flags and therefore incorrectly execute or not execute.

Conditions

The erratum requires the following sequence of instructions in ARM-state:

- VMRS<c> APSR_nzcv, FPSCR (formerly FMSTAT<c>), where the condition on the instruction is not ALways. This instruction immediately following:
- A flag-setting integer multiply or multiply and accumulate instruction (e.g. MULS), which follows:
- A single-precision floating-point multiply-accumulate (FP-MAC) instruction (e.g. VMLA), timed such that the accumulate operation is inserted into the pipeline in the cycle in which the VMRS instruction is first attempted to be issued.

Or the following instructions in Thumb-state:

- IT<c> immediately followed by VMRS APSR_nzcv, FPSCR (formerly FMSTAT), where the condition on the IT instruction is not ALways. These instructions immediately following:
- A flag-setting integer multiply instruction (e.g. MULS), which follows:
- A single-precision floating-point multiply-accumulate (FP-MAC) instruction (e.g. VMLA), timed such that the accumulate operation is inserted into the pipeline in the cycle in which the VMRS instruction is first attempted to be issued.

To meet the above timing requirements, the VMRS instruction must be three pipeline stages behind the FP-MAC. Depending on the rate in which the instructions are fetched, interlocks within this sequence and dual-issuing, this can be up to three other instructions between the this pair, plus the multiply.

This errata is unlikely to occur in practice as the code sequences are very unusual. The sequences are not generated by either the ARM or Gnu tool-chains and so will not be present in any current or legacy compiled library or application code based on these tools.

Implications

If this erratum occurs, the VMRS instruction will pass or fail its condition codes incorrectly, and this will appear in any trace produced by the ETM. This can in turn corrupt the N, Z, C, V flag values in the CPSR which will typically affect the program flow.

Workaround

There is no workaround for this erratum.

745374: Link Register may not be updated for Jazelle DBX Exception if VFP/NEON instruction is Outstanding**Category C****Products Affected: Cortex-A5 Uni-Processor.****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

756269: Watchpointed access in a store-multiple is not masked**Category C****Products Affected: Cortex-A5 Uni-Processor.****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.

756274: A load from normal memory marked with a debug watchpoint may result in an external abort**Category C****Products Affected: Cortex-A5 Uni-Processor.****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.

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

769220: ID_PFR0.state2 is incorrect when the processor includes only the trivial Jazelle implementation**Category C****Products Affected: Cortex-A5 Uni-Processor.****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.

769870: Instruction fetches from out-of-date page mappings might be observable after a TLB invalidate operation completes**Category C****Products Affected: Cortex-A5 Uni-Processor.****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, 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) The processor writes to the translation tables to modify the translation that maps virtual address VA1 to physical address PA1
- 2) The processor executes a CP15 TLB maintenance instruction that invalidates the mapping of VA1. This could be a TLBIALLIS, TLBIALL, TLBIASIDIS, TLBIASID, TLBIMVAAIS or TLBIMVAA to the same address, or TLBIMVAIS or TLBIMVA to the same address and the same ASID.
- 3) The processor executes a DSB instruction. It might execute other instructions between the TLB invalidate and the DSB.
- 4) The memory at the physical address PA1 that was previously accessible from virtual address VA1 is altered.
- 5) The processor starts fetching instructions from address VA1. This could be a speculative prefetch. Note that this instruction fetch must start before the DSB has completed, which limits the number of instructions between the DSB and the fetch from VA1 to a maximum of 17 ARM or 25 Thumb instructions.
- 6) The instruction fetch both:
 - Takes longer to complete than all outstanding data loads and stores that are in progress at the time the TLB invalidate operation is executed
 - Does not complete until after the memory at address PA1 has been altered and therefore reads the altered value.

Implications

It is not expected that software would alter the translation table mappings while executing code from those mappings at the same time on the same processor. If it does then it should already be

including an ISB instruction after the TLB invalidate. Therefore the only implications of this erratum are that incorrect data could be speculatively fetched and placed in the instruction cache. However for the data to be incorrect, the software must have modified the data at PA1, and therefore it would already need to invalidate the instruction cache for that address to ensure correct operation. If the software already does this invalidation 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 completion of the TLB invalidate operation, but already includes an ISB and instruction cache invalidate as part of the routine to map the old address out and 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

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

791672: A Non-Cacheable store in a tight loop might not become observable until the loop completes**Category C****Products Affected: Cortex-A5 Uni-Processor.****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 location. 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 uniprocessor

Conditions

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 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 processor. This will cause the stores to become observable.
- The last store will become observable to other 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 processor 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.

803371: Reads of the DBGPCSR can cause incorrect values to be latched into DBGCIDSR**Category C****Products Affected: Cortex-A5 Uni-Processor.****Present in: r0p0, r0p1****Description**

The Cortex-A5 processor supports sample-based profiling of software. Reads of the DBGPCSR register return the address of a recently-executed instruction, and cause information about the context of the instruction to be latched into DBGCIDSR. Because of this erratum, in some cases the information captured in the DBGCIDSR can be incorrect.

Configurations Affected

All configurations.

Conditions

- 1) The processor is executing in a state where non-invasive debug is permitted.
- 2) The processor transitions between Secure and Non-Secure state.
- 3) Before the processor executes the first instruction after this transition, the DBGPCSR is read.

If these conditions are met, then the values sampled into the DBGCIDSR will reflect the new context of the processor, not the context in which the sampled instruction was executed.

If the processor transitioned from Non-Secure to Secure state and non-invasive debug is not permitted in Secure state, and the DBGPCSR read is one cycle after the transition, then the DBGCIDSR will be sampled with the value of the secure CONTEXTIDR.

Implications

In cases where sample-based profiling is in use, this erratum can reduce the accuracy of the data.

If Secure non-invasive debug is not permitted, then there is a one-cycle window where a Non-secure read could observe the value of the secure CONTEXTIDR. It is not possible for reads issued by the same processor to observe this value, only another processor or agent in the system can perform a read with the required timing.

Workaround

There is no workaround for this erratum.

803570: TBB/TBH incorrect branch address is not mis-predicted**Category C****Products Affected: Cortex-A5 Uni-Processor.****Present in: r0p0, r0p1****Description**

The TBB and TBH instructions cause a PC-relative forward branch using a table of single byte or halfword offsets. A base register provides a pointer to the table, and a second register supplies an index into the table. Because of this erratum, the processor might incorrectly calculate the branch offset used to determine whether a predicted branch is correct. In this event, a branch with an address mis-prediction might be treated as a correct branch and cause the processor to execute code from the wrong address.

Configurations Affected

All configurations.

Conditions

- 1) A load or store instruction with pre-indexed base-register writeback is issued.
- 2) A TBB or TBH instruction is issued in the cycle after the load or store instruction with the same base-register as the preceding load or store instruction.
- 3) The TBB or TBH instruction is predicted taken.
- 4) The load or store instruction and the TBB or TBH instruction pass their condition code checks.
- 5) The predicted target address is equal to the base-register plus the offset loaded from memory.

If the above conditions are met then the processor branches to the base-register plus offset rather than the PC plus offset.

Example code sequence:

```
LDR r8, [r0, #0x4]!  
<independent instruction>  
TBB [r0, r5]
```

Where <independent instruction> can be one of:

- No instruction
- NOP
- IT
- B

Implications

The processor will execute code from the wrong address if permissions are correct.

The erratum will not cause the processor to change privilege level or security state.

Note: These implications can be safely ignored for compiled C/C++ code because the branch table is opaque to the programmer and C/C++ compilers will not generate the code sequence described in conditions (1) to (4). Similarly, the code sequence is not expected to be generated by other compilers or interpreters. In addition, the 32-bit result of the calculation in condition (5) is unlikely to match the branch target address predicted by the BTAC. This erratum has not been observed in real code.

Workaround

There is no workaround for this erratum.

807269: PMU event counters 0x07, 0x0C, 0x0E and 0x14 do not increment correctly**Category C****Products Affected: Cortex-A5 Uni-Processor.****Present in: r0p0, r0p1****Description**

The Cortex-A5 processor implements version 2 of the Performance Monitor Unit architecture (PMUv2). The PMU can gather statistics on the operation of the processor and memory system during runtime. This event information can be used when debugging or profiling code.

The PMU can be programmed to count architecturally executed stores (event 0x07), software changes of the PC (event 0x0C), procedure returns (event 0x0E) and L1 instruction cache accesses (event 0x14). However, because of this erratum, these events do not fully adhere to the descriptions in the PMUv2 architecture.

Configurations Affected

All configurations.

Conditions**Either:**

- 1) A PMU counter is enabled and programmed to count event 0x07. That is: instruction architecturally executed, condition code check pass, store.
- 2) A PLDW instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x07 does not increment. However, the counter does increment.

Or:

- 1) A PMU counter is enabled and programmed to count event 0x0C. That is: instruction architecturally executed, condition code check pass, software change of the PC.
- 2) An SVC or SMC instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0C increments. However, the counter does not increment.

Or:

- 1) A PMU counter is enabled and programmed to count event 0x0E. That is: instruction architecturally executed, condition code check pass, procedure return.
- 2) One of the following instructions is executed:
 - a. MOV PC, LR
 - b. ThumbEE LDMIA R9!, {..., PC}
 - c. ThumbEE LDR PC, [R9], #offset
 - d. BX Rm, where Rm != R14
 - e. LDM SP, {..., PC}

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0E increments for (a), (b), (c) and does not increment for (d) and (e). However, the counter does not increment for (a), (b), (c) and increments for (d) and (e).

Or:

- 1) A PMU counter is enabled and programmed to count event 0x14. That is: L1 instruction cache accesses.
- 2) Cacheable instruction fetches miss in the L1 instruction cache.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x14 increments. However, the counter does not increment.

Implications

The information returned by PMU counters that are programmed to count events 0x07, 0x0C, 0x0E, or 0x14 might be misleading when debugging or profiling code executed on the processor.

Workaround

There is no workaround for PMU events 0x7, 0xC and 0xE.

To obtain a better approximation for the number of L1 instruction cache accesses, enable a second PMU counter and program it to count instruction fetches that cause linefills (event 0x01). Add the value returned by this counter to the value returned by the L1 instruction access counter (event 0x14). The result of the addition is a better indication of the number of L1 instruction cache accesses.