ARM Processor
Cortex™-M4 (AT520) and Cortex-M4 with FPU (AT521)

Product Revision r0

Software Developers Errata Notice

Non-Confidential - Released

ARM®

Software Developers Errata Notice

Copyright © 2013 ARM. 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-039104
- 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.

This is the first version of the Software Developers Errata Notice document for Cortex-M4 products. Please note that ARM has previously published one of the two errata in this document. An erratum identified since the previous publication is listed in the change record below.

01 Apr 2013: Changes in Document v1

Page	Status	ID	Cat	Rare	Summary of Erratum
8	New	776924	CatB		VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

Contents

Chapter 1.
Introduction

This chapter introduces the errata notice for the ARM processors Cortex-M4 and Cortex-M4 with Floating Point Unit.

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 $\boxed{\textbf{X}}$ 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
776924	CatB		VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	X	X
752770	CatB		Interrupted loads to SP can cause erroneous behaviour	X	X

2.3. Category A

There are no errata in this category

2.4. Category A (Rare)

There are no errata in this category

2.5. Category B

752770: Interrupted loads to SP can cause erroneous behaviour

Category B
Products Affected: Cortex-M4, Cortex-M4F.
Present in: r0p0, r0p1

Description

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), erroneous behaviour can occur. In all cases, returning from the interrupt will result in the load instruction being executed an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The affected instructions that can result in the load transaction being repeated are:

1) LDR SP,[Rn],#imm

2) LDR SP,[Rn,#imm]!

3) LDR SP,[Rn,#imm]

4) LDR SP,[Rn]

5) LDR SP,[Rn,Rm]

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

1) LDR SP,[Rn],#imm

2) LDR SP,[Rn,#imm]!

Conditions

1) An LDR is executed, with SP/R13 as the destination

2) The address for the LDR is successfully issued to the memory system

3) An interrupt is taken before the data has been returned and written to the stack-pointer.

Implications

Unless the load is being performed to Device or Strongly-Ordered memory, there should be no implications from the repetition of the load. In the unlikely event that the load is being performed to Device or Strongly-Ordered memory, the repeated read can result in the final stack-pointer value being different than had only a single load been performed.

Interruption of the two write-back forms of the instruction can result in both the base register value and final stack-pointer value being incorrect. This can result in apparent stack corruption and subsequent unintended modification of memory.

Workaround

Both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

If repeated reads are acceptable, then the base-update issue may be worked around by performing the stack-pointer load without the base increment followed by a subsequent ADD or SUB instruction to perform the appropriate update to the base register.

[776924](#): VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

Category B
Products Affected: Cortex-M4F.
Present in: r0p0, r0p1

Description

On Cortex-M4 with FPU, the VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to the FPSCR.

Conditions

1) The floating point unit is present and enabled

2) Lazy context saving is not disabled

3) A VDIV or VSQRT is executed

4) The destination register for the VDIV or VSQRT is one of s0 - s15

5) An interrupt occurs and is taken

6) The interrupt service routine being executed does not contain a floating point instruction

7) 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

A minimum of 12 of these 14 cycles are utilised for the context state stacking, which leaves 2 cycles for instructions inside the interrupt service routine, or 2 wait states applied to the entire stacking sequence (which means that it is not a constant wait state for every access).

In general this means that if the memory system inserts wait states for stack transactions then this erratum cannot be observed.

Implications

The VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, meaning that these registers hold incorrect, out of date, data.

Workaround

A workaround is only required if the floating point unit is present and enabled. A workaround is not required if the memory system inserts one or more wait states to every stack transaction.

There are two workarounds:

1) Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).

2) Ensure that every interrupt service routine contains more than 2 instructions in addition to the exception return instruction.

2.6. Category B (Rare)

There are no errata in this category

2.7. Category C

There are no errata in this category