Cortex-A9 MPCore™

Programmer Advice Notice

Read-after-Read Hazards

ARM Reference 761319

ARM®

Cortex-A9 MPCore

Programmer Advice Notice

Read-after-Read Hazards

ARM Reference 761319

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Confidentiality	Change
22 September 2011	A	Non-Confidential	First release

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Feedback on Content

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

- the title
- the document number, ARM UAN 0004A
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Web Address

`http://www.arm.com`

Table of Contents

1. Preface

1.1 Intended audience

This document is intended for programmers and compiler developers who deploy a software workaround for a possible issue, identified as ARM Reference 761319 on Cortex-A9 MPCore processors.

1.2 Document status

This document is final, that is for a developed product.

1.3 References

This document refers to the following documents:

References

Document number or reference	Title
http://en.wikipedia.org/wiki/Non-blocking_algorithm	Non-blocking algorithms
ARM DDI 0406	ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition

1.4 Terms and abbreviations

This document uses the following terms and abbreviations:

Terms and abbreviations

Term	Description
Lock-free programming	A multi-threaded programming methodology that avoids the use of lock variables when communicating between threads of execution.
Volatile (storage class qualifier)	In C and C++ this is the `volatile` storage class qualifier. In other languages the syntax and semantics might vary slightly if the concept is supported at all. The intent is to cover all storage locations that might be used for inter-processor communication variables that can be used for lock-free programming.

2. Introduction

This Programmer Advice Notice describes a problem that can occur, in rare circumstances, when performing successive reads from the same memory location on a Cortex-A9 MPCore processor.

This document describes changes that can be applied to toolchains. These changes avoid the problem in any code generated for and executed on a Cortex-A9 MPCore processor. However, depending on the design of the toolchain, applying these suggestions might not be possible.

3. Problem description

3.1 Processors affected

This problem affects all revisions of the Cortex-A9 MPCore processor.

3.2 Instruction sequences affected

On all versions of the Cortex-A9 MPCore processor, in very rare circumstances, successive reads from the same location in Normal Write-Back Shared memory that is being modified by another processor can result in the read values not appearing in program order.

3.3 Example code

3.3.1 Example of unsafe code

The following code might be unsafe:

On processor 1

```
STR   <valueA>, [loc]
…
STR   <valueB>, [loc]
```

On processor 2

```
LDR   Rx, [loc]
…                   // No barriers
LDR   Ry, [loc]
```

The result of executing these code sequences is that processor 2 might occasionally incorrectly observe `Rx == <valueB>` and `Ry == <valueA>`.

The problem can affect all forms of memory load instruction except `LDREX`, `LDREXB`, `LDREXH` and `LDREXD`.

For high-level languages, compilers often optimize away or otherwise re-order multiple accesses to the same memory location, so only memory locations that are declared volatile (see *Volatile (storage class qualifier)* in *Terms and abbreviations* on page 4) can be considered to be susceptible to this problem. Furthermore, any barrier operation between the two loads is sufficient to prevent the conditions for triggering the problem, so only lock-free programming methodologies are affected (see *Non-blocking algorithms* in *References* on page 4).

3.3.2 Examples of safe code

The following code is safe:

On processor 2

```
LDR   Rx, [loc]
DMB
…
LDR   Ry, [loc]
DMB
```

Or, on processor 2

```
LDREX Rx, [loc]
…
LDREX Ry, [loc]
```

Note: the use of `LDREX` is UNPREDICTABLE unless accessing Normal memory, so the second sequence cannot be used as a general workaround for accessing all volatile objects in memory.

4. Workaround

It is impractical to work around this problem in a linker because the number of load instructions is too large to make binary patching feasible. In addition, the majority of load instructions can never be affected by this problem because they are never used for inter-thread communication, but a linker cannot distinguish the different classes of access. Instead compiler developers must implement the following workaround and programmers must recompile affected code with the workaround enabled.

Programmers must manually fix hand-written assembly language and inline assembly instructions in source code.

The workaround is to intercept all volatile memory reads in the compiler and issue a `DMB` instruction immediately afterwards. This ensures that all such accesses are ordered correctly.

It is possible that device drivers will have their performance materially affected by this solution, so ARM recommends that this workaround is only enabled as a result of a command-line option.