Cortex™-A9 NEON™ Media Processing Engine

Revision: r4p1

Technical Reference Manual

ARM DDI 0409I (ID091612)

Cortex-A9 NEON Media Processing Engine
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
04 April 2008	A	Non-Confidential	First release for r0p0
10 July 2008	B	Non-Confidential Restricted Access	Second release for r0p0
12 Dec 2008	C	Non-Confidential Restricted Access	First release for r1p0
24 September 2009	D	Non-Confidential Restricted Access	First release for r2p0
27 November 2009	E	Non-Confidential Unrestricted Access	Second release for r2p0
27 April 2010	F	Non-Confidential Unrestricted Access	First release for r2p2
21 July 2011	G	Non-Confidential	First release for r3p0
26 March 2012	H	Non-Confidential	First release for r4p0
15 June 2012	I	Non-Confidential	First release for r4p1

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

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.

Web Address

`http://www.arm.com`

Contents
Cortex-A9 NEON Media Processing Engine Technical Reference Manual

Preface

This preface introduces the *Cortex-A9 NEON™ Media Processing Engine (MPE) Technical Reference Manual*. It contains the following sections:

About this book

This book is for the Cortex-A9 NEON MPE.

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.

Intended audience

This book is written for system designers, system integrators, and verification engineers who are designing a *System-on-Chip* (SoC) device that uses the Cortex-A9 NEON MPE. The book describes the external functionality of the Cortex-A9 MPE.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the Cortex-A9 implementation of the ARM Advanced SIMD media processing architecture.

Chapter 2 *Programmers Model*

Read this for a description of the Cortex-A9 NEON programmers model.

Chapter 3 *Instruction Timing*

Read this for a description of the cycle timings of instructions on the Cortex-A9 NEON MPE.

Appendix A *Revisions*

Read this for a description of the technical changes between released issues of this book.

Glossary

The *ARM Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM Glossary*, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Conventions

Conventions that this book can use are described in:

- *Typographical conventions*

Typographical conventions

The typographical conventions are:

italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Applies when the relevant term is used in body text. For example: IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, `http://infocenter.arm.com`, for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *Cortex-A9 Technical Reference Manual* (ARM DDI 0388)
- *Cortex-A9 MPCore Technical Reference Manual* (ARM DDI 0407)
- *Cortex-A9 Floating-Point Unit Technical Reference Manual* (ARM DDI 0408)
- *Cortex-A9 MBIST Controller Technical Reference Manual* (ARM DDI 0414)
- *Cortex-A9 Configuration and Sign-Off Guide* (ARM DII 0146)
- *CoreSight™ PTM™-A9 Technical Reference Manual* (ARM DDI 0401)
- *CoreSight PTM-A9 Configuration and Sign-Off Guide* (ARM DII 0161)
- *CoreSight PTM-A9 Integration Manual* (ARM DII 0162)
- *CoreSight Program Flow Trace Architecture Specification* (ARM IHI 0035)
- *AMBA® Level 2 Cache Controller (L2C-310) Technical Reference Manual* (ARM DDI 0246)
- *L220 Cache Controller Technical Reference Manual* (ARM DDI 0329)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *AMBA Specification* (ARM IHI 0011)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *RealView™ Compilation Tools Developer Guide* (ARM DUI 0203)
- *RealView ICE and RealView Trace User Guide* (ARM DUI 0155)

- *Intelligent Energy Controller Technical Overview* (ARM DTO 0005).

Other publications

This section lists relevant documents published by third parties:

- *ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic*.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

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

- the title
- the number, ARM DDI 0409I
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

This chapter introduces the Cortex-A9 implementation of the ARM Advanced SIMD media processing architecture. It contains the following sections:

- *About the Cortex-A9 NEON MPE* on page 1-2
- *Applications* on page 1-4
- *Product revisions* on page 1-5.

1.1 About the Cortex-A9 NEON MPE

The Cortex-A9 NEON MPE extends the Cortex-A9 functionality to provide support for the ARM v7 *Advanced SIMD* and *Vector Floating-Point v3* (VFPv3) instruction sets. The Cortex-A9 NEON MPE supports all addressing modes and data-processing operations described in the *ARM Architecture Reference Manual*.

The Cortex-A9 NEON MPE features are:

- SIMD and scalar single-precision floating-point computation
- scalar double-precision floating-point computation
- SIMD and scalar half-precision floating-point conversion
- 8, 16, 32, and 64-bit signed and unsigned integer SIMD computation
- 8 or 16-bit polynomial computation for single-bit coefficients
- structured data load capabilities
- dual issue with Cortex-A9 processor ARM or Thumb instructions
- independent pipelines for VFPv3 and Advanced SIMD instructions
- large, shared register file, addressable as:
    - thirty-two 32-bit S (single) registers
    - thirty-two 64-bit D (double) registers
    - sixteen 128-bit Q (quad) registers.

  See the *ARM Architecture Reference Manual* for details of the information that the MPE can hold in the different register formats.

The Cortex-A9 NEON MPE provides high-performance SIMD vector operations for:

- unsigned and signed integers
- single bit coefficient polynomials
- single-precision floating-point values.

The operations include:

- addition and subtraction
- multiplication with optional accumulation
- maximum or minimum value driven lane selection operations
- inverse square-root approximation
- comprehensive data-structure load instructions, including register-bank-resident table lookup.

See the *ARM Architecture Reference Manual* for details of the Advanced SIMD instructions.

Note

The Advanced SIMD architecture extension, its associated implementations, and supporting software, are commonly referred to as NEON™ technology.

1.1.1 VFPv3 architecture hardware support

The Cortex-A9 NEON MPE hardware supports single and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations as described in the ARM VFPv3 architecture. It provides conversions between 16-bit, 32-bit and 64-bit floating-point formats and ARM integer word formats, with special operations to perform conversions in round-towards-zero mode for high-level language support.

All instructions are available in both the ARM and Thumb instruction sets supported by the Cortex-A9 processor family.

The Cortex-A9 NEON MPE provides an optimized solution in performance, power, and area for embedded and media intensive applications.

ARMv7 deprecates the use of VFP vector mode. The Cortex-A9 NEON MPE hardware does not support VFP vector operations. In this manual, the term vector refers to Advanced SIMD integer, polynomial and single-precision vector operations. The Cortex-A9 NEON MPE provides high speed VFP operation without support code. However, if an application requires VFP vector operation, then it must use support code. See the *ARM Architecture Reference Manual* for information on VFP vector operation support.

Note

This manual gives information specific to the Cortex-A9 NEON MPE implementation of the ARM Advanced SIMD and VFPv3 extensions. See the *ARM Architecture Reference Manual* for full instruction set and usage details.

1.2 Applications

The Cortex-A9 NEON MPE provides mixed-data type SIMD and high-performance scalar floating-point computation suitable for a wide spectrum of applications such as:

- personal digital assistants and smartphones for graphics, voice compression and decompression, user interfaces, Java interpretation, and *Just In Time* (JIT) compilation
- games machines for intensive three-dimensional graphics, digital audio and in-game physics effects such as gravity
- printers and *MultiFunction Peripheral* (MFP) controllers for high-definition color rendering
- set-top boxes for high-end digital audio and digital video, and interactive three-dimensional user interfaces
- automotive applications for engine management, power train computation, and in-car entertainment and navigation.

1.3 Product revisions

This section describes the differences in functionality between product revisions:

r0p0 - r1p0	There are no functionality changes although you must use the Cortex-A9 revision r1p0 design with revision r1p0 NEON MPE.
r1p0 - r2p0	There are no functionality changes although you must use the Cortex-A9 revision r2p0 design with this revision r2p0 NEON MPE.
r2p0 - r2p1	There are no functionality changes although you must use the Cortex-A9 revision r2p1 design with this revision r2p1 NEON MPE.
r2p1 - r2p2	There are no functionality changes.
r2p2 - r3p0	There are no functionality changes.
r3p0 - r4p0	There are no functionality changes.
r4p0 - r4p1	There are no functionality changes.

Chapter 2
Programmers Model

This chapter describes the Cortex-A9 NEON MPE programmers model. It contains the following sections:

2.1 About this programmers model

This section introduces the VFPv3 and Advanced SIMD implementation provided by the Cortex-A9 NEON MPE. In addition it provides information on initializing the Cortex-A9 NEON MPE ready for application code execution. These are described in:

- *Advanced SIMD and VFP feature identification registers*
- *Enabling Advanced SIMD and floating-point support*.

In addition to features provided in previous combined Advanced SIMD and VFP implementations, the Cortex-A9 NEON MPE provides:

- half-precision, 16-bit, floating-point value conversion
- support for emulation of VFPv3-D32 and VFPv3-D16.

See the *ARM Architecture Reference Manual* for more information.

2.1.1 Advanced SIMD and VFP feature identification registers

The Cortex-A9 NEON MPE implements the ARMv7 Advanced SIMD and VFP extensions.

Software can identify these extensions and the features they provide, using the feature identification registers. The extensions are in the coprocessor space for coprocessors CP10 and CP11. You can access the registers using the `VMRS` and `VMSR` instructions, for example:

```
VMRS <Rd>, FPSID ; Read Floating-Point System ID Register
VMRS <Rd>, MVFR1 ; Read Media and VFP Feature Register 1
VMSR FPSCR, <Rt> ; Write Floating-Point System Control Register
```

See *Advanced SIMD and VFP register access* on page 2-6 for a description of the registers.

In addition there are coprocessor access control registers. See *Non-secure Access Control Register* on page 2-7 and *Coprocessor Access Control Register* on page 2-6.

2.1.2 Enabling Advanced SIMD and floating-point support

From reset, both the Advanced SIMD and VFP extensions are disabled. Any attempt to execute either a NEON or VFP instruction results in an Undefined Instruction exception being taken. To enable software to access Advanced SIMD and VFP features ensure that:

- Access to CP10 and CP11 is enabled for the appropriate privilege level. See *Coprocessor Access Control Register* on page 2-6.

- If Non-secure access to the Advanced SIMD features or VFP features is required, the access flags for CP10 and CP11 in the NSACR must be set to 1. See *Non-secure Access Control Register* on page 2-7.

In addition, software must set the FPEXC.EN bit to 1 to enable most Advanced SIMD and VFP operations. See *Floating-Point Exception Register* on page 2-13.

When Advanced SIMD and VFP operation is disabled because FPEXC.EN is 0, all Advanced SIMD and VFP instructions are treated as undefined instructions except for execution of the following in privileged modes:

- a VMSR to the FPEXC or FPSID register
- a VMRS from the FPEXC, FPSID, MVFR0 or MVFR1 registers.

Example 2-1 on page 2-3 shows how to enable the Advanced SIMD and VFP in ARM *Unified Assembly Language* (UAL). This code must be executed in privileged mode.

Example 2-1 Enabling Advanced SIMD and VFP

```
MRC  p15,0,r0,c1,c0,2 ; Read CPACR into r0
ORR  r0,r0,#(3<<20)   ; OR in User and Privileged access for CP10
ORR  r0,r0,#(3<<22)   ; OR in User and Privileged access for CP11
BIC  r0, r0, #(3<<30) ; Clear ASEDIS/D32DIS if set
MCR  p15,0,r0,c1,c0,2 ; Store new access permissions into CPACR
ISB                   ; Ensure side-effect of CPACR is visible
MOV  r0,#(1<<30)      ; Create value with FPEXC (bit 30) set in r0
VMSR FPEXC,r0         ; Enable VFP and SIMD extensions
```

At this point the Cortex-A9 processor can execute Advanced SIMD and VFP instructions.

Note

Operation is UNPREDICTABLE if you configure the *Coprocessor Access Control Register* (CPACR) so that CP10 and CP11 do not have identical access permissions.

2.2 New Advanced SIMD and VFP features

The Cortex-A9 NEON MPE implements the following new features in the ARMv7 Advanced SIMD and VFP architectures:

- *Half-precision floating-point value conversion*
- *Independent Advanced SIMD and VFP disable*
- *Dynamically configurable VFP register bank size*.

Full details of the new instructions and control register fields are in the *ARM Architecture Reference Manual*.

2.2.1 Half-precision floating-point value conversion

The half-precision floating-point value conversion adds support for both IEEE and the common graphic representation, referred to as alternative-half-precision representation, of 16-bit floating-point values. This provides a smaller memory footprint for applications requiring large numbers of lower-precision floating-point values to be stored, while avoiding the overhead of conversion in software.

Additional VFP and Advanced SIMD instructions enable conversion of both individual values and vectors of values to and from single-precision floating-point representation. These values can then be processed using the rest of the VFP and Advanced SIMD instructions.

See *Floating-Point Status and Control Register* on page 2-11 for how to select IEEE or alternative half-precision modes.

2.2.2 Independent Advanced SIMD and VFP disable

The independent Advanced SIMD disables permit Cortex-A9 implementations with Cortex-A9 NEON MPE to behave as though only the VFP extension were present. This lets you enable optimal operating-system task scheduling between Cortex-A9 multi-processor clusters containing both Cortex-A9 NEON MPE and floating-point only units.

The Cortex-A9 processor provides support for preventing use of this feature through Non-secure access. See *Non-secure Access Control Register* on page 2-7 and *Coprocessor Access Control Register* on page 2-6.

2.2.3 Dynamically configurable VFP register bank size

The dynamically configurable VFP register bank size provides additional support for both VFPv3-D16 and VFPv3-D32 mixed multiprocessor clusters. Cortex-A9 NEON MPE implements thirty-two 64-bit double-precision registers. VFP-only implementations are only required to support sixteen double-precision registers. This register bank disable control enables emulation of a 16-entry double-precision register file, providing both enhanced compatibility and more flexible task scheduling.

Additional control is provided in the Non-Secure Access Control Register. See *Non-secure Access Control Register* on page 2-7 and *Coprocessor Access Control Register* on page 2-6.

2.3 Supported formats

Table 2-1 shows the formats supported for each of the Advanced SIMD and VFPv3 instruction sets implemented by the Cortex-A9 NEON MPE. All signed integers are two's complement representations.

Table 2-1 Supported number formats

Format	Advanced SIMD	VFPv3
8-bit signed/unsigned integer	Yes	No
16-bit signed/unsigned integer	Yes	No
32-bit signed/unsigned integer	Yes	Yes[a]
64-bit signed/unsigned integer	Yes	No
16-bit half-precision floating-point	Yes[a]	Yes[a]
32-bit single-precision floating-point	Yes	Yes
64-bit double-precision floating-point	No	Yes
8-bit polynomials	Yes	No
16-bit polynomials	Yes	No

a. For conversion purposes only.

2.4 Advanced SIMD and VFP register access

Table 2-2 shows the system control coprocessor registers, accessed through CP15, that determine access to Advanced SIMD and VFP registers, where:

- CRn is the register number within CP15
- Op1 is the Opcode_1 value for the register
- CRm is the operational register
- Op2 is the Opcode_2 value for the register.

Table 2-2 Coprocessor Access Control registers

CRn	Op1	CRm	Op2	Name	Description
c1	0	c0	2	CPACR	See *Coprocessor Access Control Register*
c1	0	c1	2	NSACR	See *Non-secure Access Control Register* on page 2-7

2.4.1 Coprocessor Access Control Register

The CPACR Register sets access rights for the coprocessors CP10 and CP11, that enable the Cortex-A9 NEON MPE functionality. This register also enables software to determine if a particular coprocessor exists in the system.

The CPACR Register is:

- a read/write register common to Secure and Non-secure states
- accessible in privileged modes only
- has a reset value of 0.

Figure 2-1 shows the CPACR Register bit assignments.

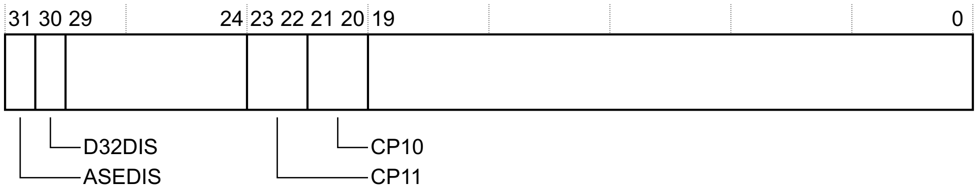

Figure 2-1 CPACR Register bit assignments

Table 2-3 shows the CPACR Register bit assignments.

Table 2-3 Coprocessor Access Control Register bit assignments

Bits	Field	Function
[31]	ASEDIS	Disable Advanced SIMD extension functionality: **0** No instructions are disabled. **1** Disables all instruction encodings identified in the *ARM Architecture Reference Manual* as being part of the Advanced SIMD extensions but that are not VFPv3 instructions.
[30]	D32DIS	Disable use of D16-D31 of the VFP register file: **0** No instructions are disabled. **1** Disables all instruction encodings identified in the *ARM Architecture Reference Manual* as being VFPv3 instructions if they access any of registers D16-D31.
[29:24]	-	See the *Cortex-A9 Technical Reference Manual*.

Table 2-3 Coprocessor Access Control Register bit assignments (continued)

Bits	Field	Function
[23:22]	CP11	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for nonexistent coprocessors. `b00` Access denied. Attempted access generates an Undefined Instruction exception. `b01` Privileged mode access only. `b10` Reserved. `b11` Privileged and User mode access.
[21:20]	CP10	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for nonexistent coprocessors. `b00` Access denied. Attempted access generates an Undefined Instruction exception. `b01` Privileged mode access only. `b10` Reserved. `b11` Privileged and User mode access.
[19:0]	-	See the *Cortex-A9 Technical Reference Manual*.

Access to coprocessors in the Non-secure state depends on the permissions set in the *Non-secure Access Control Register*.

Attempts to read or write the CPACR Register access bits depend on the corresponding bit for each coprocessor in *Non-secure Access Control Register*. Table 2-4 shows the results of attempted access to coprocessor access bits for each mode.

Table 2-4 Results of access to the CRACR Register

NSACR[11:10]	Secure privileged	Non-secure privileged	Secure or Non-secure User
b00	R/W	RAZ/WI	Access prohibited[a]
b01	R/W	R/W	Access prohibited[a]

[a] User privilege access generates an Undefined Instruction exception.

To access the CPACR Register, read or write CP15 with:

```
MRC p15, 0,<Rd>, c1, c0, 2 ; Read Coprocessor Access Control Register
MCR p15, 0,<Rd>, c1, c0, 2 ; Write Coprocessor Access Control Register
```

When the CPACR is updated, the change to the register is guaranteed to be visible only after the next *Instruction Synchronization Barrier* (ISB) instruction. When this register is updated, software must ensure that no instruction that depends on the new or old register values is issued before the `ISB` instruction.

Normally, software uses a read, modify, write sequence to update the CPACR, to avoid unwanted changes to the access settings for other coprocessors.

Note

You must enable CP10 and CP11 in the CPACR Register before accessing any Advanced SIMD or VFP system registers.

2.4.2 Non-secure Access Control Register

The NSACR Register defines the Non-secure access rights for the Cortex-A9 NEON MPE and other system functionality.

The NSACR Register is:

- a read/write register in Secure state
- a read-only register in Non-secure state
- only accessible in privileged modes.

Figure 2-2 shows the bit assignments of the NSACR Register relevant to the Cortex-A9 MPE. See the *Cortex-A9 Technical Reference Manual* for details of other fields in this register.

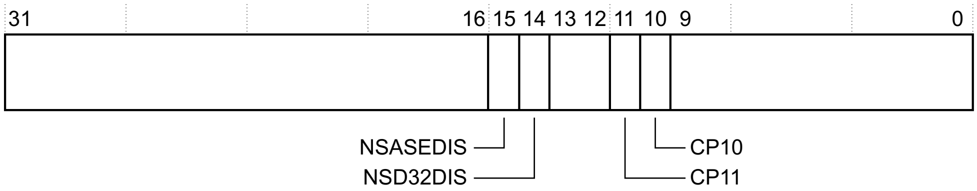

Figure 2-2 NSACR Register bit assignments

Table 2-5 shows the NSACR Register bit assignments.

Table 2-5 NSACR Register bit assignments

Bits	Field	Function
[31:16]	-	See the *Cortex-A9 Technical Reference Manual*.
[15]	NSASEDIS	Disable Non-secure Advanced SIMD extension functionality: **0** Full access provided to CPACR.ASEDIS. **1** The CPACR.ASEDIS bit when executing in Non-secure state has a fixed value of 1 and writes to it are ignored.
[14]	NSD32DIS	Disable Non-secure use of D16-D31 of the VFP register file: **0** Full access provided to CPACR.D32DIS. **1** The CPACR.D32DIS bit when executing in Non-secure state has a fixed value of 1 and writes to it are ignored.
[13:12]	-	See the *Cortex-A9 Technical Reference Manual*.
[11]	CP11	Permission to access coprocessor 11: **0** Secure access only. This is the reset value. **1** Secure or Non-secure access.
[10]	CP10	Permission to access coprocessor 10: **0** Secure access only. This is the reset value. **1** Secure or Non-secure access.
[9:0]	-	See the *Cortex-A9 Technical Reference Manual*.

To access the NSACR Register, read or write CP15 with:

```
MRC p15, 0,<Rd>, c1, c1, 2 ; Read Non-secure Access Control Register data
MCR p15, 0,<Rd>, c1, c1, 2 ; Write Non-secure Access Control Register data
```

Table 2-6 on page 2-9 shows the results of attempted access for each mode.

Table 2-6 Results of access to the NSACR Register

Secure privileged		Non-secure privileged		User	
Read	**Write**	**Read**	**Write**	**Read**	**Write**
Data	Data	Data	Undefined Instruction exception	Undefined Instruction exception	Undefined Instruction exception

2.5 Register summary

Table 2-7 shows the Cortex-A9 NEON MPE system registers. All NEON MPE system registers are 32-bit wide. Reserved register addresses are RAZ/WI.

Table 2-7 Cortex-A9 NEON MPE system registers

Name	Type	Reset	Description
FPSID	RO	`0x41033094`	See *Floating-Point System ID Register* on page 2-11
FPSCR	RW	`0x00000000`	See *Floating-Point Status and Control Register* on page 2-11
MVFR1	RO	`0x01111111`	See the *ARM Architecture Reference Manual*
MVFR0	RO	`0x10110222`	See the *ARM Architecture Reference Manual*
FPEXC	RW	`0x00000000`	See *Floating-Point Exception Register* on page 2-13

Table 2-8 shows the processor modes for accessing the Cortex-A9 NEON MPE system registers.

Table 2-8 Accessing Cortex-A9 NEON MPE system registers

Register	Privileged access		User access	
	FPEXC EN=0	FPEXC EN=1	FPEXC EN=0	FPEXC EN=1
FPSID	Permitted	Permitted	Not permitted	Not permitted
FPSCR	Not permitted	Permitted	Not permitted	Permitted
MVFR0, MVFR1	Permitted	Permitted	Not permitted	Not permitted
FPEXC	Permitted	Permitted	Not permitted	Not permitted

2.6 Register descriptions

This section describes the Cortex-A9 NEON MPE system registers. Table 2-7 on page 2-10 provides cross references to individual registers.

2.6.1 Floating-Point System ID Register

The FPSID Register characteristics are:

Purpose	Provides information about the VFP implementation.
Usage constraints	Only accessible in privileged modes.
Configurations	Available in all NEON MPE configurations.
Attributes	See the register summary in Table 2-7 on page 2-10.

Figure 2-3 shows the FPSID Register bit assignments.

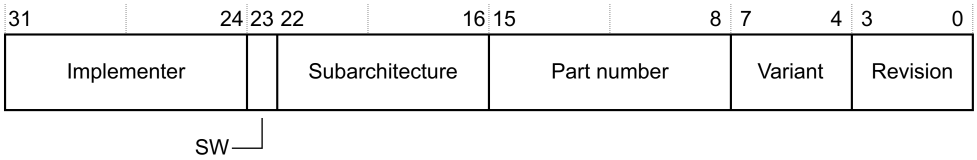

Figure 2-3 FPSID Register bit assignments

Table 2-9 shows the FPSID Register bit assignments.

Table 2-9 FPSID Register bit assignments

Bits	Name	Function
[31:24]	Implementer	Denotes ARM
[23]	SW	Hardware implementation with no software emulation
[22:16]	Subarchitecture	The null VFP sub-architecture
[15:8]	Part number	VFPv3
[7:4]	Variant	Cortex-A9
[3:0]	Revision	Revision 4

You can access the FPSID Register with the following `VMRS` instruction:

```
VMRS <Rd>, FPSID ; Read Floating-Point System ID Register
```

2.6.2 Floating-Point Status and Control Register

The FPSCR characteristics are:

Purpose	Provides User level control of the FPU.
Usage constraints	There are no usage constraints.
Configurations	Available in all FPU configurations.
Attributes	See the register summary in Table 2-7 on page 2-10.

Figure 2-4 on page 2-12 shows the FPSCR bit assignments.

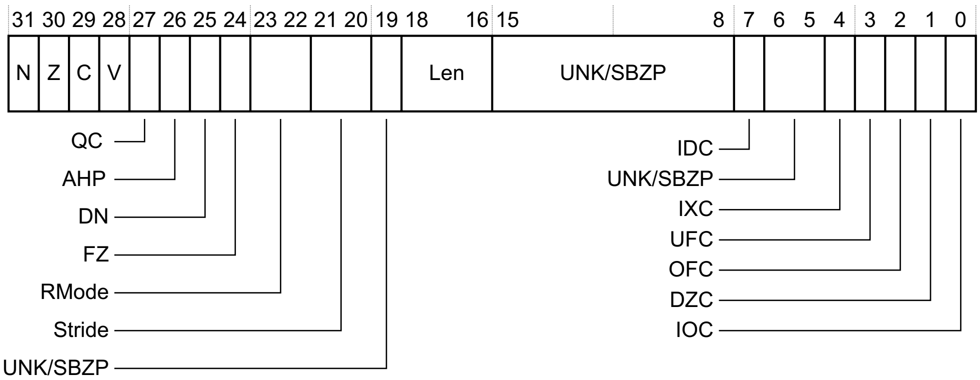

Figure 2-4 FPSCR bit assignments

Table 2-10 shows the FPSCR bit assignments.

Table 2-10 FPSCR bit assignments

Bits	Field	Function
[31]	N	Set to 1 if a comparison operation produces a less than result.
[30]	Z	Set to 1 if a comparison operation produces an equal result.
[29]	C	Set to 1 if a comparison operation produces an equal, greater than, or unordered result.
[28]	V	Set to 1 if a comparison operation produces an unordered result.
[27]	QC	Set to 1 if an Advanced SIMD integer operation has saturated since 0 was last written to this bit.[a]
[26]	AHP	Alternative Half-Precision control bit: **0** IEEE half-precision format selected. **1** Alternative half-precision format selected.
[25]	DN	Default NaN mode control bit: **0** NaN operands propagate through to the output of a floating-point operation **1** Any operation involving one or more NaNs returns the Default NaN. Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.
[24]	FZ	Flush-to-zero mode control bit: **0** Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. **1** Flush-to-zero mode enabled. Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.
[23:22]	RMode	Rounding Mode control field: `b00` *Round to Nearest* (RN) mode `b01` *Round towards Plus infinity* (RP) mode `b10` *Round towards Minus infinity* (RM) mode `b11` *Round towards Zero* (RZ) mode. Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.
[21:20]	Stride	Stride control used for backwards compatibility with short vector operations. The Cortex-A9 NEON MPE ignores the value of this field. See the *ARM Architecture Reference Manual*.
[19]	-	UNK/SBZP.

Table 2-10 FPSCR bit assignments (continued)

Bits	Field	Function
[18:16]	Len	Vector length, used for backwards compatibility with short vector operation. If you set this field to a non-zero value, the VFP data-processing instructions generate exceptions. See the *ARM Architecture Reference Manual*.
[15:8]	-	UNK/SBZP.
[7]	IDC	Input Denormal cumulative exception flag.[a]
[6:5]	-	UNK/SBZP.
[4]	IXC	Inexact cumulative exception flag.[a]
[3]	UFC	Underflow cumulative exception flag.[a]
[2]	OFC	Overflow cumulative exception flag.[a]
[1]	DZC	Division by Zero cumulative exception flag.[a]
[0]	IOC	Invalid Operation cumulative exception flag.[a]

[a] The exception flags, bit [27], bit [7], and bits [4:0] of the FPSCR are exported on the **DEFLAGS** output so they can be monitored externally to the processor, if required.

You can access the FPSCR with the following `VMSR` instructions:

```
VMRS <Rd>, FPSCR ; Read Floating-Point Status and Control Register
VMSR FPSCR, <Rt> ; Write Floating-Point Status and Control Register
```

2.6.3 Floating-Point Exception Register

The FPEXC Register characteristics are:

Purpose Provides global enable control of the Advanced SIMD and VFP extensions.

Usage constraints

- Only accessible in the Non-secure state if the CP10 and CP11 bits in the NSACR are set to 1, see *Non-secure Access Control Register* on page 2-7.
- Only accessible in privileged modes, and only if access to coprocessors CP10 and CP11 is enabled in the Coprocessor Access Control Register, see *Coprocessor Access Control Register* on page 2-6.

Configurations Available in all NEON MPE configurations.

Attributes See the register summary in Table 2-7 on page 2-10.

Figure 2-5 on page 2-14 shows the FPEXC Register bit assignments.

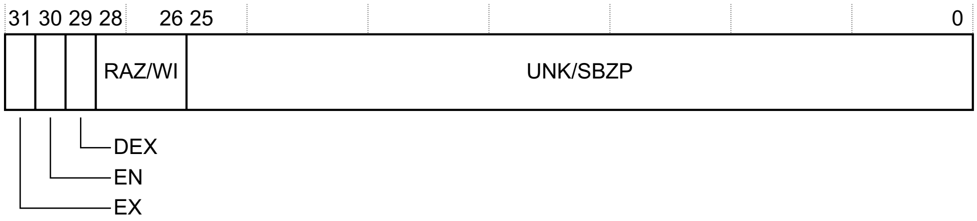

Figure 2-5 FPEXC Register bit assignments

Table 2-11 shows the FPEXC Register bit assignments.

Table 2-11 FPEXC Register bit assignments

Bits	Name	Function
[31]	EX	Exception bit: This bit reads-as-zero and ignores-writes. The Cortex-A9 NEON MPE never requires asynchronous exception handling.
[30]	EN	Enable bit: **0** The Advanced SIMD and VFP extensions are disabled. **1** The Advanced SIMD and VFP extensions are enabled and operate normally. The EN bit is cleared to 0 at reset.
[29]	DEX	Defined synchronous instruction exceptional flag: **0** No exception has occurred. **1** Attempt to perform a VFP vector operation has been trapped[a]. The DEX bit is cleared to 0 at reset.
[28:26]	-	RAZ/WI.
[25:0]	-	UNK/SBZP.

[a] The Cortex-A9 NEON MPE hardware does not support the deprecated VFP short vector feature. Attempts to execute VFP data-processing instructions when the FPSCR.LEN field is non-zero result in the FPCSR.DEX bit being set and a synchronous Undefined instruction exception being taken. You can use software to emulate the short vector feature, if required.

You can access the FPEXC Register with the following `VMSR` instructions:

```
VMRS <Rd>, FPEXC ; Read Floating-Point Status and Control Register
VMSR FPEXC, <Rt> ; Write Floating-Point Status and Control Register
```

Chapter 3
Instruction Timing

This chapter describes the cycle timings of instructions on the Cortex-A9 NEON MPE. It contains the following sections:

- *About instruction cycle timing* on page 3-2
- *Writing optimal VFP and Advanced SIMD code* on page 3-3
- *Cortex-A9 NEON MPE instructions* on page 3-4
- *Instruction-specific scheduling* on page 3-9.

3.1 About instruction cycle timing

This chapter provides information to estimate how much execution time particular code sequences require. The complexity of the Cortex-A9 processor makes it impossible to guarantee precise timing information with hand calculations. The timing of an instruction is often affected by other concurrent instructions, memory system activity, and events outside the instruction flow.

Detailed descriptions of all possible instruction interactions and events taking place in the processor are beyond the scope of this document.

3.2 Writing optimal VFP and Advanced SIMD code

The following guidelines can provide significant performance increases for VFP and Advanced SIMD code:

Where possible avoid:

- unnecessary accesses to the VFP control registers
- transferring values between the Cortex-A9 core registers and VFP or Advanced SIMD register file, see the *ARM Architecture Reference Manual* for definition of core registers
- register dependencies between neighboring instructions
- mixing Advanced SIMD only instructions with VFP only instructions.

Be aware that:

- with the exception of simultaneous loads and stores, the processor can execute VFP and Advanced SIMD instructions in parallel with ARM or Thumb instructions
- using Advanced SIMD value selection operations is more efficient than using the equivalent VFP compare with conditional execution.

3.3 Cortex-A9 NEON MPE instructions

Table 3-1 shows the instructions supported by the Cortex-A9 NEON MPE, and the instruction set that they are in, either Advanced SIMD or VFP. The data types that each instruction supports are as follows:

D	Double precision floating-point values
F	Single precision floating-point values
H	Half precision floating-point values
I	Integer values
P	Polynomials with single-bit coefficients
X	Operation is independent of data representation.

See the *ARM Architecture Reference Manual* for details of instruction encodings and functionality.

Table 3-1 Cortex-A9 MPE instructions

Name	Advanced SIMD	VFP	Description
`VABA`	I	-	Absolute Difference and Accumulate
`VABAL`	I	-	Absolute Difference and Accumulate Long
`VABD`	I, F	-	Absolute Difference
`VABDL`	I	-	Absolute Difference Long
`VABS`	I, F	F, D	Absolute
`VACGE`	F	-	Absolute Compare Greater Than or Equal
`VACGT`	F	-	Absolute Compare Greater Than
`VACLE`	F	-	Absolute Compare Less Than or Equal
`VACLT`	F	-	Absolute Compare Less Than
`VADD`	I, F	F, D	Add
`VADDHN`	I	-	Add and Narrow Returning High Half
`VADDL`	I	-	Add Long
`VADDW`	I	-	Add Wide
`VAND`	X	-	Bitwise AND
`VBIC`	I	-	Bitwise Clear
`VBIF`	X	-	Bitwise Insert if False
`VBIT`	X	-	Bitwise Insert if True
`VBSL`	X	-	Bitwise Select
`VCEQ`	I, F	-	Compare Equal
`VCGE`	I, F	-	Compare Greater Than or Equal
`VCLE`	I, F	-	Compare Less Than or Equal
`VCLS`	I	-	Count Leading Sign Bits
`VCLT`	I, F	-	Compare Less Than

Table 3-1 Cortex-A9 MPE instructions (continued)

Name	Advanced SIMD	VFP	Description
`VCLZ`	I	-	Count Leading Zeros
`VCMP`	-	F, D	Compare Setting Flags
`VCNT`	I	-	Count Number of Set Bits
`VCVT`	I, F, H	I, F, D, H	Convert Between Floating-Point and 32-bit Integer Types
`VDIV`	-	F, D	Divide
`VDUP`	X	-	Duplicate
`VEOR`	X	-	Bitwise Exclusive OR
`VEXT`	I	-	Extract Elements and Concatenate
`VHADD`	I	-	Halving Add
`VHSUB`	I	-	Halving Subtract
`VLD1`	X	-	Load Single-Element Structures
`VLD2`	X	-	Load Two-Element Structures
`VLD3`	X	-	Load Three-Element Structures
`VLD4`	X	-	Load Four-Element Structures
`VLDM`	X	F, D	Load Multiple Registers
`VLDR`	X	F, D	Load Single Register
`VMAX`	I, F	-	Maximum
`VMIN`	I, F	-	Minimum
`VMLA`	I, F	F, D	Multiply Accumulate
`VMLS`	I, F	F, D	Multiply Subtract
`VMLAL`	I	-	Multiply Accumulate Long
`VMLSL`	I	-	Multiply Subtract Long
`VMOV`	X	F, D	Move Register or Immediate
`VMOVL`	I	-	Move Long
`VMOVN`	I	-	Move and Narrow
`VMRS`	X	F, D	Move Advanced SIMD or VFP Register to ARM Compute Engine
`VMSR`	X	F, D	Move ARM Core Register to Advanced SIMD or VFP
`VMUL`	I, F, P	F, D	Multiply
`VMULL`	I, F, P	-	Multiply Long
`VMVN`	X	-	Bitwise NOT
`VNEG`	I, F	F, D	Negate
`VNMLA`	-	F, D	Negative Multiply Accumulate
`VNMLS`	-	F, D	Negative Multiply Subtract

Table 3-1 Cortex-A9 MPE instructions (continued)

Name	Advanced SIMD	VFP	Description
`VNMUL`	-	F, D	Negative Multiply
`VORN`	X	-	Bitwise OR NOT
`VORR`	X	-	Bitwise OR
`VPADAL`	I	-	Pairwise Add and Accumulate Long
`VPADD`	I, F	-	Pairwise Add
`VPADDL`	I	-	Pairwise Add Long
`VPMAX`	I, F	-	Pairwise Maximum
`VPMIN`	I, F	-	Pairwise Minimum
`VPOP`	X	F, D	Pop from Stack
`VPUSH`	X	F, D	Push to Stack
`VQABS`	I	-	Saturating Absolute
`VQADD`	I	-	Saturating Add
`VQDMLAL`	I	-	Saturating Double Multiply Accumulate Long
`VQDMLSL`	I	-	Saturating Double Multiply Subtract Long
`VQDMULH`	I	-	Saturating Doubling Multiply Returning High Half
`VQDMULL`	I	-	Saturating Doubling Multiply Long
`VQMOVN`	I	-	Saturating Move and Narrow
`VQMOVUN`	I	-	Saturating Move and Unsigned Narrow
`VQNEG`	I	-	Saturating Negate
`VQRDMULH`	I	-	Saturating Rounding Doubling Multiply Returning High Half
`VQRSHL`	I	-	Saturating Rounding Shift Left
`VQRSHRN`	I	-	Saturating Rounding Shift Right Narrow
`VQRSHRUN`	I	-	Saturating Rounding Shift Right Unsigned Narrow
`VQSHL`	I	-	Saturating Shift Left
`VQSHLU`	I	-	Saturating Shift Left Unsigned
`VQSHRN`	I	-	Saturating Shift Right Narrow
`VQSHRUN`	I	-	Saturating Shift Right Unsigned Narrow
`VQSUB`	I	-	Saturating Subtract
`VRADDHN`	I	-	Rounding Add and Narrow Returning High Half
`VRECPE`	I, F	-	Reciprocal Estimate
`VRECPS`	F	-	Reciprocal Step
`VREV16`	X	-	Reverse in Halfwords
`VREV32`	X	-	Reverse in Words

Table 3-1 Cortex-A9 MPE instructions (continued)

Name	Advanced SIMD	VFP	Description
`VREV64`	X	-	Reverse in Doublewords
`VRHADD`	I	-	Rounding Halving Add
`VRSHL`	I	-	Rounding Shift Left
`VRSHR`	I	-	Rounding Shift Right
`VRSHRN`	I	-	Rounding Shift Right Narrow
`VRSQRTE`	I, F	-	Reciprocal Square Root Estimate
`VRSQRTS`	F	-	Reciprocal Square Root Step
`VRSRA`	I	-	Rounding Shift Right and Accumulate
`VRSUBHN`	I	-	Rounding Subtract and Narrow Returning High Half
`VSHL`	I	-	Shift Left
`VSHLL`	I	-	Shift Left Long
`VSHR`	I	-	Shift Right
`VSHRN`	I	-	Shift Right Narrow
`VSLI`	X	-	Shift Left and Insert
`VSQRT`	-	F, D	Square Root
`VSRA`	I	-	Shift Right and Accumulate
`VSRI`	X	-	Shift Right and Insert
`VST1`	X	-	Store single-element structures
`VST2`	X	-	Store two-element structures
`VST3`	X	-	Store three-element structures
`VST4`	X	-	Store four-element structures
`VSTM`	X	F, D	Store Multiple Registers
`VSTR`	X	F, D	Store Register
`VSUB`	I, F	F, D	Subtract
`VSUBHN`	I	-	Subtract and Narrow
`VSUBL`	I	-	Subtract Long
`VSUBW`	I	-	Subtract Wide
`VSWP`	I	-	Swap Contents
`VTBL`	X	-	Table Lookup
`VTBX`	X	-	Table Extension
`VTRN`	X	-	Transpose

Table 3-1 Cortex-A9 MPE instructions (continued)

Name	Advanced SIMD	VFP	Description
`VTST`	I	-	Test Bits
`VUZP`	X	-	Unzip
`VZIP`	X	-	Zip

3.4 Instruction-specific scheduling

Complex instruction dependencies and memory system interactions make it impossible to concisely specify the exact cycle timing of all instructions in all circumstances. The following sections provide timing information that is accurate in most cases:

- *Instruction timing tables*
- *VFP instruction timing* on page 3-10
- *VFP load and store instruction timing* on page 3-11
- *Advanced SIMD integer arithmetic instructions* on page 3-12
- *Advanced SIMD integer multiply instructions* on page 3-14
- *Advanced SIMD integer shift instructions* on page 3-15
- *Advanced SIMD permute instructions* on page 3-16
- *Advanced SIMD floating-point instructions* on page 3-17
- *Advanced SIMD load-store instructions* on page 3-18.
- *Inter-pipeline data-transfers* on page 3-19.

For precise timing, you must use a cycle-accurate model of both your system and the processor.

3.4.1 Instruction timing tables

The tables in this section provide useful timing information for each category of instruction. The definition of each column is as follows:

Instruction Instruction mnemonics are in the ARM UAL format. For legacy mnemonic equivalents, see the *ARM Architecture Reference Manual*.

Format This is the register format of the instruction for the timing information that is provided. Where multiple formats are given in a single row, the timing information applies to all those formats. In some cases, the table provides operand type information when different operands require different computation time. For example, Table 3-2 on page 3-10, the VFP timing table, contains .F and .D UAL modifiers for single and double-precision operations respectively. Where data-type specifiers are omitted, then you can assume the information applies to all specifiers relevant to the particular entry.

Cycles This is the number of issue cycles the particular instruction consumes, and is the absolute minimum number of cycles per instruction if no operand interlocks are present.

Source The Source field indicates the execution cycle where the source operands must be available if the instruction is to be permitted to issue. The comma separated list matches that of the Format field, indicating the register that each value applies to.

Where two lines are provided for a single format containing quad (Q) registers, this indicates that the source registers are handled independently between top and bottom half double (D) registers. The first line provides the information for the lower half of the quad register and the second line provides the same information for the upper half of the quad register.

Result The Result field indicates the execution cycle when the result of the operation is ready. At this point, the result might be ready as source operands for consumption by other instructions using forwarding paths. However, some pairs of instructions might have to wait until the value is written back to the register file.

Writeback The Writeback field indicates the execution cycle that the result is committed to the register file. From this cycle, the result of the instruction is available to all other instructions as a source operand.

Note

An instruction might not issue if its writeback to a particular register might occur before that of an earlier instruction to the same register.

This section contains the following:

- *VFP instruction timing*
- *Advanced SIMD integer arithmetic instructions* on page 3-12
- *Advanced SIMD integer multiply instructions* on page 3-14
- *Advanced SIMD integer shift instructions* on page 3-15
- *Advanced SIMD permute instructions* on page 3-16
- *Advanced SIMD floating-point instructions* on page 3-17
- *Advanced SIMD load-store instructions* on page 3-18.

3.4.2 VFP instruction timing

Table 3-2 shows the VFP instruction timing.

Table 3-2 VFP instruction timing

Name	Format	Cycles	Source	Result	Writeback
`VADD` `VSUB`	`.F Sd,Sn,Sm` `.D Dd,Dn,Dm`	1	-,1,1	4	4
`VCVT`	`.F Sd,Sn` `.D Dd,Sn`	1	-,1	4	4
`VMUL`	`.F Sd,Sn,Sm`	1	-,1,1	5	5
`VNMUL`	`.D Dd,Dn,Dm`	2	-,1,1	6	6
`VMLA`[a]	`.F Sd,Sn,Sm`	1	-,1,1	8	8

`VMLS`
`VNMLS`
`VNMLA`
	`.D Dd,Dn,Dm`	2	-,1,1	9	9

`VABS` `VNEG`	`.F Sd,Sn` `.D Dd,Dn`	1	-,1	1	2
`VMOV`[b]	`Rt,Sn`	1	-,1	-	-
	`Rt,Rt2,Dn`	1	-,-,1	-	-
	`Dd,Rt,Rt2`	1	-,1,1	1	2
	`Sd,RtSd,SnDd,Dn`	1	-,1	1	2
	`.F Sd,#imm` `.D Dd,#imm`	1	-,-	1	2
`VMRS`	`Rt,FPSCR`	1	-,1	-	-
`VDIV`	`.F Sd,Sn,Sm`	10	-,1,1	15	15
	`.D Dd,Dn,Dm`	20	-,1,1	25	25

Table 3-2 VFP instruction timing (continued)

Name	Format	Cycles	Source	Result	Writeback
`VSQRT`	`.F Sd, Sm`	13	-,1,1	17	17
	`.D Dd, Dm`	28	-,1,1	32	32
`VCMP`	`.F Sn,Sm` `.D Dn,Dm`	1	1,1	1	4

[a] If a multiply-accumulate follows a multiply or another multiply-accumulate, and depends on the result of that first instruction, then if the dependency between both instructions are of the same type and size, the processor uses a special multiplier accumulator forwarding. This special forwarding means the multiply instructions can issue back-to-back because the result of the first instruction in cycle 5 is forwarded to the accumulator of the second instruction in cycle 4. If the size and type of the instructions do not match, then Dd or Qd is required in cycle 3. This applies to combinations of the multiply-accumulate instructions `VMLA`, `VMLS`, `VQDMLA`, and `VQDMLS`, and the multiply instructions `VMUL` and`VQDMUL`.

[b] The numbers in this table represent the source, result and writeback values from n MPE point of view only. See *Inter-pipeline data-transfers* on page 3-19 for timing information about data transfers between the MPE and the ARM register file.

3.4.3 VFP load and store instruction timing

Table 3-3 shows the VFP load/store instruction timing.

The SP registers column gives the number of single-precision registers that are operated on. For load and store multiple operations, the number of registers in the list is equal to either 2p or 2p+1 for an even or odd number of registers respectively. Load and stores operations using addresses that are not 64-bit aligned can incur an additional cycle.

Table 3-3 VFP load and store instruction timing

Name	Format	64-bit aligned	SP registers	Cycles	Source[a]	Result	Writeback
`VLDR`	`.F Sd,[]`	-	1	1	-	1	1
`VLDR`	`.D Dd,[]`	Yes	2	1	-	1	1
		No	2	2	-	1,2	1,2
`VLDM`	`.F Rn,{...}`	Yes	2p	p	-	1, 2, …, p	1, 2, …, p
			2p+1	p+1	-	1, 2, …, p+1	1, 2, …, p+1
		No	2p	p+1	-	1, 2, …, p+1	1, 2, …, p+1
			2p+1	p+1	-	1, 2, …, p+1	1, 2, …, p+1
`VLDM`	`.D Rn,{...}`	Yes	2p	p	-	1, 2, …, p	1, 2, …, p
		No	2p	p+1	-	1, 2, …, p+1	1, 2, …, p+1
`VSTR`	`.F Sd,[]`	-	1	1	-	-	-
`VSTR`	`.D Dd,[]`	Yes	2	1	-	-	-
		No	2	2	-	-	-

Table 3-3 VFP load and store instruction timing (continued)

Name	Format	64-bit aligned	SP registers	Cycles	Source[a]	Result	Writeback
`VSTM`	`.F Rn,{...}`	Yes	2p	p	-	-	-
			2p+1	p+1	-	-	-
		No	2p	p+1	-	-	-
			2p+1	p+1	-	-	-
`VSTM`	`.D Rn,{...}`	Yes	2p	p	-	-	-
		No	2p	p+1	-	-	-

[a] Store instructions can be issued before their operands are calculated.

3.4.4 Advanced SIMD integer arithmetic instructions

Table 3-4 shows the Advanced SIMD integer arithmetic instruction timing.

Table 3-4 Advanced SIMD integer arithmetic instruction timing

Name	Format	Cycles	Source	Result	Writeback

`VADD`
`VAND`
`VORR`
`VEOR`
`VBIC`
`VORN`
	`Dd,Dn,Dm`
	`Qd,Qn,Qm`
		1	-,2,2	3	6

`VSUB`	`Dd,Dn,Dm` `Qd,Qn,Qm`	1	-,2,1	3	6
`VADDL` `VSUBL`	`Qd,Dn,Dm`	1	-,1,1	3	6
`VADDW` `VSUBW`	`Qd,Qn,Dm`	1	-,2,1	3	6

`VHADD`
`VRHADD`
`VQADD`
`VTST`
	`Dd,Dn,Dm`
	`Qd,Qn,Qm`
		1	-,2,2	4	6

`VADH` `VRADH`	`Dd,Qn,Qm`	1	-,2,2	4	6
`VSBH` `VRSBH`	`Dd,Qn,Qm`	1	-,2,1	4	6

Table 3-4 Advanced SIMD integer arithmetic instruction timing (continued)

Name	Format	Cycles	Source	Result	Writeback
VHSUB VQSUB VABD VCEQ VCGE VCGT VMAX VMIN	Dd,Dn,Dm Qd,Qn,Qm	1	-,2,1	4	6
VPMAX VPMIN	Dd,Dn,Dm	1	-,2,1	4	6
VNEG	Dd,Dm Qd,Qm	1	-,1	3	6
VQNEG VQABS	Dd,Dm Qd,Qm	1	-,1	4	6
VABDL	Qd,Dn,Dm	1	-,2,1	4	6
VABS	Dd,Dm Qd,Qm	1	-,2	4	6
VCEQ VCGE VCGT VCLE VCLT	Dd,Dm,#imm Qd,Qm,#imm	1	-,2,-	4	6
VPADD	Dd,Dn,Dm	1	-,1,1	3	6
VPADDL	Dd,Dn Qd,Qn	1	-,1	3	6
VMVN	Dd,Dm, Qd,Qm	1	-,2	3	6
VCLS VCLZ VCNT	Dd,Dm	1	-,2	3	6
	Qd,Qm	2	-,2 -,3	3 4	6 7
VMOV VMVN	Dd,#imm Qd,#imm	1	-,-	3	6
VORR VBIC	Dd,#imm Qd,#imm	1	2,-	3	6
VBIT VBIF VBSL	Dd,Dn,Dm	1	2,2,2	3	6
	Qd,Qn,Qm	2	2,2,2 3,3,3	3 4	6 7

Table 3-4 Advanced SIMD integer arithmetic instruction timing (continued)

Name	Format	Cycles	Source	Result	Writeback
VABA	Dd,Dn,Dm	1	3,2,1	6	6
	Qd,Qn,Qm	2	3,2,1 4,3,2	6	6
VABAL	Qd,Dn,Dm	1	3,2,1	6	6
VPADAL	Dd,Dm Qd,Qm	1	3,1	6	6

3.4.5 Advanced SIMD integer multiply instructions

Table 3-5 shows the operation of the Advanced SIMD integer multiply instruction timing.

Table 3-5 Advanced SIMD integer multiply instructions

Name	Format	Cycles	Source	Result	Writeback
VMUL VQDMLH VQRDMLH	.8 Dd,Dn,Dm .16 Dd,Dn,Dm	1	-,2,2	6	6
	.32 Dd,Dn,Dm	2	-,2,1	7	7
	.32 Qd,Qn,Qm	4	-,2,1 -,4,3	7 9	7 9
VMULL VQDMULL	.8 Qd,Dn,Dm .16 Qd,Dn,Dm	1	-,2,2	6	6
	.32 Qd,Dn,Dm	2	-,2,1	7	7
VMLA VMLS	.8 Dd,Dn,Dm .16 Dd,Dn,Dm	1	3,2,2	6	6
	.8 Qd,Qn,Qm .16 Qd,Qn,Qm	2	3,2,2 4,3,3	6 7	6 7
	.32 Dd,Dn,Dm	2	3,2,1	7	7
	.32 Qd,Qn,Qm	4	3,2,1 5,4,3	7 9	7 9
VMLAL VMLSL VQDMLAL VQDMLSL	.8 Qd,Dn,Dm .16 Qd,Dn,Dm	1	3,2,2	6	6
	.32 Qd,Dn,Dm	2	3,2,1	7	7
VMUL VQDMLH VQRDMLH	.16 Dd,Dn,Dm[x]	1	-,2,1	6	6
	.16 Qd,Qn,Dm[x]	2	-,2,1 -,3,1	6 7	6 7
	.32 Dd,Dn,Dm[x]	2	-,2,1	7	7
	.32 Qd,Qn,Qm[x]	4	-,2,1 -,4,1	7 9	7 9

Table 3-5 Advanced SIMD integer multiply instructions (continued)

Name	Format	Cycles	Source	Result	Writeback
VMULL VQDMULL	.16 Qd,Dn,Dm[x]	1	-,2,1	6	6
	.32 Qd,Dn,Dm[x]	2	-,2,1	7	7
VMLA VMLS	.16 Dd,Dn,Dm[x]	1	3,2,1	6	6
	.16 Qd,Qn,Dm[x]	2	3,2,1	6	6
	.32 Dd,Dn,Dm[x]	2	3,2,1	7	7
	.32 Qd,Qn,Dm[x]	4	3,2,1 5,2,1	7 9	7 9
VMLAL VMLSL VQDMLAL VQDMLSL	.16 Qd,Dn,Dm[x]	1	3,2,1	6	6
	.32 Qd,Dn,Dm[x]	2	3,2,1	7	7

3.4.6 Advanced SIMD integer shift instructions

Table 3-6 shows the Advanced SIMD integer shift instruction timing.

Table 3-6 Advanced SIMD integer shift instruction timing

Name	Format	Cycles	Source	Result	Writeback
VSHR VSHL	Dd,Dm,#imm Qd,Qm,#imm	1	-,1,-	3	6
VQSHL VRSHR	Dd,Dm,#imm Qd,Qm,#imm	1	-,1,-	4	6
VSHRN VMOVN	Dd,Qm,#imm	1	-,1,-	3	6
VQSHRN VQMOVN VQSHRN VQRSHR	Dd,Qm,#imm	1	-,1,-	4	6
VSHL	Qd,Dm,#imm	1	-,1,-	3	6
VMOVL	Qd,Dm	1	-,1	3	6
VSLI VSRI	Dd,Dm,#imm	1	1,1,-	3	6
	Qd,Qm,#imm	2	1,1,- 2,2,-	3 4	6 7
VSHL	Dd,Dm,Dn	1	-,1,1	3	6
	Qd,Qm,Qn	2	-,1,1 -,2,2	3 4	6 7

Table 3-6 Advanced SIMD integer shift instruction timing (continued)

Name	Format	Cycles	Source	Result	Writeback

`VQSHL`
`VRSHL`
`VQRSHL`
	`Dd,Dm,Dn`	1	-,1,1	4	6

	`Qd,Qm,Qn`	2	-,1,1 -,2,2	5	7
`VSRA`	`Dd,Dm,#imm`	1	3,1,-	4	6
`VRSRA`	`Qd,Qm,#imm`	1	3,1,-	6	6

3.4.7 Advanced SIMD permute instructions

Table 3-7 shows the operation of the Advanced SIMD permute instruction timing.

Table 3-7 Advanced SIMD permute instruction timing

Name	Format	Cycles	Source	Result	Writeback
`VDUP` `VTRN`	`Dd,Dm[x]` `Qd,Qm[x]`	1	-,1	2	6
`VSWP`	`Dd,Dm`	1	1,1	2	6
	`Qd,Qm`	2	1,1 2,2	2 3	6 7
`VZIP`	`Dd,Dm`	1	1,1	2	6
	`Qd,Qm`	3	1,1 2,2	3,4	7,8
`VUZP`	`Dd,Dm`	1	1,1	2	6
	`Qd,Qm`	3	1,2	3	7
`VREV`	`Dd,DmQd,Qm`	1	-,1	2	6
`VEXT`	`Dd,Dn,Dm,#imm`	1	-,1,1,-	2	6
	`Qd,Qn,Qm,#imm`	2	-,1,2,-	3	7
`VTBL`	`Dd,{Dn},Dm`	2	-,2,1	3	7
	`Dd,{Dn,Dn1},Dm`	2	-,2,2,1	3	7
	`Dd,{Dn,Dn1,Dn2},Dm`	3	-,2,2,3,1	4	8
	`Dd,{Dn,Dn1,Dn2,Dn3},Dm`	3	-,2,2,3,3,1	4	8
`VTBX`	`Dd,{Dn},Dm`	2	1,2,1	3	7
	`Dd,{Dn,Dn1},Dm`	2	1,2,2,1	3	7
	`Dd,{Dn,Dn1,Dn2},Dm`	3	1,2,2,3,1	4	8
	`Dd,{Dn,Dn1,Dn2,Dn3},Dm`	3	1,2,2,3,3,1	4	8

3.4.8 Advanced SIMD floating-point instructions

Table 3-8 shows the Advanced SIMD floating-point instructions.

Table 3-8 Advanced SIMD floating-point instructions

Name	Format	Cycles	Source	Result	Writeback

`VADD`
`VSUB`
`VABD`
`VMUL`
`VCEQ`
`VCGE`
`VCGT`
`VACGE`
`VACGT`
`VMAX`
`VMIN`
	`Dd,Dn,Dm`	1	-,2,2	5	6

	`Qd,Qn,Qm`	2	-,2,2 -,3,3	5 6	6 7

`VABS`
`VNEG`
`VRECPE`
`VRSQRTE`
`VCVT`
	`Dd,Dm`	1	-,2	5	6

	`Qd,Qm`	2	-,2 -,3	5 6	6 7

`VCEQ`
`VCGE`
`VCGT`
`VCLE`
`VCLT`
	`Dd,Dm,#imm`	1	-,2	5	6

	`Qd,Qm,#imm`	2	-,2 -,3	5 6	6 7

`VPADD`
`VPMAX`
`VPMIN`
	`Dd,Dn,Dm`	1	-,1,1	5	6

`VMUL`	`Dd,Dn,Dm[x]`	1	-,2,1	5	6
	`Qd,Qn,Dm[x]`	2	-,2,1 -,3,1	5 6	6 7
`VMLA` `VMLS`	`Dd,Dn,Dm`	1	3,2,2	9	10
	`Qd,Qn,Qm`	2	3,2,2 4,3,3	9 10	10 11
	`Dd,Dn,Dm[x]`	1	3,2,1	9	10
	`Qd,Qn,Dm[x]`	2	3,2,1 4,3,1	9 10	10 11
`VRECPS` `VRSQRTS`	`Dd,Dn,Dm`	1	-,2,2	9	10
	`Qd,Qm,Qn`	2	-,2,2 -,3,3	9 10	10 11

3.4.9 Advanced SIMD load-store instructions

Table 3-9 shows the Advanced SIMD load-store instruction timing.

The values in Table 3-9 correspond to the number of issue cycles required within the MPE execution unit. The number of cycles required by the Cortex-A9 integer processor is equal to the number of 64-bit aligned doublewords that the NEON load or store data overlaps.

Table 3-9 Advanced SIMD load-store instructions

Name	Format	Cycles	Source	Result	Writeback
`VLD1`	`{Dd},[]`	2	-	2	7
	`{Dd},[@]`	1	-	1	6
	`{Dd,Dd1},[]`	2	-,-	2,2	7,7
	`{Dd,Dd1},[@]`	1	-,-	1,1	6,6
	`{Dd,Dd1,Dd2},[]`	3	-,-,-	2,2,3	7,7,8
	`{Dd,Dd1,Dd2},[@]`	2	-,-,-	1,1,2	6,6,7
	`{Dd,Dd1,Dd2,Dd3},[]`	3	-,-,-,-	2,2,3,3	7,7,8,8
	`{Dd,Dd1,Dd2,Dd3},[@]`	2	-,-,-,-	1,1,2,2	6,6,7,7
	`{Dd[x]},[]`	3	1	4	8
	`{Dd[x]},[@]`	2	1	3	7
	`{Dd[]},[]`	2	-	3	7
	`{Dd[]},[@]`	1	-	2	6
	`{Dd[],Dd1[]},[]`	2	-,-	3,3	7,7
	`{Dd[],Dd1[]},[@]`	1	-,-	2,2	6,6
`VLD2`	`{Dd,Dd1},[]`	2	-,-	3,3	7,7
	`{Dd,Dd1},[@]`	1	-,-	2,2	6,6
	`{Dd,Dd1,Dd2,Dd3},[]`	3	-,-,-,-	3,4,3,4	7,8,7,8
	`{Dd,Dd1,Dd2,Dd3},[@]`	2	-,-,-,-	2,3,2,3	6,7,6,7
	`{Dd[x],Dd1[x]},[]`	3	1,1	4,4	8,8
	`{Dd[x],Dd1[x]},[@]`	2	1,1	3,3	7,7
	`{Dd[],Dd1[]},[]`	2	-,-	3,3	7,7
	`{Dd[],Dd1[]},[@]`	1	-,-	2,2	6,6
`VLD3`	`{Dd,Dd1,Dd2},[]`	4	-,-,-	4,4,5	8,8,9
	`{Dd,Dd1,Dd2},[@]`	3	-,-,-	3,3,4	7,7,8
	`{Dd[x],…,Dd2[x]},[]`	5	1,1,2	5,5,6	9,9,10
	`{Dd[],Dd1[],Dd2[]},[]`	3	-,-,-	3,3,4	7,7,8

Table 3-9 Advanced SIMD load-store instructions (continued)

Name	Format	Cycles	Source	Result	Writeback
VLD4	{Dd,Dd1,Dd2,Dd3},[]	4	-,-,-,-	4,4,5,5	8,8,9,9
	{Dd,Dd1,Dd2,Dd3},[@]	3	-,-,-,-	3,3,4,4	7,7,8,8
	{Dd[x],…,Dd3[x]},[]	5	1,1,2,2	5,5,6,6	9,9,10,10
	{Dd[x],…,Dd3[x]},[@]	4	1,1,2,2	4,4,5,5	8,8,9,9
	{Dd[],…,Dd3[]},[]	3	-,-,-,-	3,3,4,4	7,7,8,8
	{Dd[],….,Dd3[]},[@]	2	-,-,-,-	2,2,3,3	6,6,7,7
VST1	{Dd},[]	2	1	-	-
	{Dd},[@]	1	1	-	-
	{Dd[x]},[]	2	1	-	-
	{Dd[x]},[@]	1	1	-	-
VST2	{Dd,Dd1},[]	2	1,1	-	-
	{Dd,Dd1},[@]	1	1,1	-	-
	{Dd[x],Dd1[x]},[]	2	1,1	-	-
	{Dd[x],Dd1[x]},[@]	1	1,1	-	-
VST3	{Dd,Dd1,Dd2},[]	3	1,1,2	-	-
	{Dd,Dd1,Dd2},[@]	2	1,1,2	-	-
	{Dd[x],…,Dd2[x]},[]	3	1,1,2	-	-
VST4	{Dd,Dd1,Dd2,Dd3},[]	3	1,1,2,2	-	-
	{Dd,Dd1,Dd2,Dd3},[@]	2	1,1,2,2	-	-
	{Dd[x],…,Dd3[x]},[]	3	1,1,2,2	-	-
	{Dd[x],…,Dd3[x]},[@]	2	1,1,2,2	-	-

3.4.10 Inter-pipeline data-transfers

A Cortex-A9 processor, implemented with the MPE, contains distinct data-processing units:

- the integer core
- a unit, within the MPE, for handling Advanced-SIMD instructions
- a unit, within the MPE, for handling VFP instructions

In addition to the cycles listed in the previous tables for data movement within a unit, additional latencies apply for moving data values between units. Table 3-10 shows the additional latency incurred for moving values between the three units, either using explicit move instructions, or in the case of Advanced-SIMD to or from VFP fields, where a value generated by one unit is consumed by another:

Table 3-10 Inter-unit latencies

	To		
From	**Integer Core**	**Advanced-SIMD**	**VFP**
Integer Core	-	0	0
Advanced-SIMD	11	-	-
VFP	3	-	-

The VFP and Advanced-SIMD units can not execute concurrently. All instructions targeting either the Advanced-SIMD or VFP units complete before later instructions are issued on the other unit.

Appendix A
Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Differences between issue A and issue B

Change	Location
Removed text stating that implementation of Security Extensions is optional	Throughout book
Created separate entry for the `VSWP` instruction	Table 3-7 on page 3-16
Updated values for the `VMLA`, `VMLS`, `VRECPS`, and `VRSQRTS` instructions	Table 3-8 on page 3-17

Table A-2 Differences between issue B and issue C

Change	Location
Modified the example code for enabling the Advanced SIMD and VFP extensions in ARM Unified Assembly Language	Example 2-1 on page 2-3
Updated FPSCR bit assignments table	Table 2-10 on page 2-12

Table A-3 Differences between issue C and issue D

Change	Location
Value for version updated to 2	Table 2-7 on page 2-10
FPSCR SBZ/WI bits retitled to UNK/SBZP	Table 2-10 on page 2-12 and Figure 2-4 on page 2-12
SBZ/WI bits retitled to UNK/SBZ	Table 2-11 on page 2-14 and Figure 2-5 on page 2-14

Table A-3 Differences between issue C and issue D (continued)

Change	Location
Added Footnote about special behavior of two `VMLA` instructions	Table 3-2 on page 3-10
Footnote reworded	Table 3-3 on page 3-11
Added paragraph about timings	Table 3-9 on page 3-18
Table formatting fixes	Table 3-3 on page 3-11 Table 3-4 on page 3-12 Table 3-5 on page 3-14 Table 3-6 on page 3-15 Table 3-7 on page 3-16 Table 3-8 on page 3-17 Table 3-9 on page 3-18

Table A-4 Differences between issue D and Issue E

Change	Location
No technical change	-

Table A-5 Differences between issue E and Issue F

Change	Location
ISB added to code example	Example 2-1 on page 2-3
Harmonized FPEXC bit descriptions with the FPU TRM descriptions	Table 2-11 on page 2-14

Table A-6 Differences between issue F and issue G

Change	Location	Affects
No technical change	-	-

Table A-7 Differences between issue G and issue H

Change	Location	Affects
Update Revision number	Table 2-7 on page 2-10 Table 2-9 on page 2-11	r4p0

Table A-8 Differences between issue H and Issue I

Change	Location
No technical change	-