MOVE Coprocessor

Copyright © 2001, 2002, 2004. ARM Limited. All rights reserved.

Release Information

The following changes have been made to this manual.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2001</td>
<td>A</td>
<td>First release.</td>
</tr>
<tr>
<td>April 2002</td>
<td>B</td>
<td>Input and output added to Figure 2-2. Signal name changed from DABORT to CPABORT on page A-2.</td>
</tr>
<tr>
<td>23 August 2004</td>
<td>C</td>
<td>Updated confidentiality.</td>
</tr>
</tbody>
</table>

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited, 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 Limited 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.

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

MOVE Coprocessor

Preface
About this manual ................................................................. x
Feedback ................................................................. xiv

Chapter 1 Introduction
1.1 About the MOVE coprocessor ............................................. 1-2
1.2 MOVE structure .............................................................. 1-3
1.3 MOVE interface .............................................................. 1-4

Chapter 2 Functional Description
2.1 MOVE overview .............................................................. 2-2
2.2 Connectivity ................................................................. 2-3
2.3 Functional overview ....................................................... 2-4

Chapter 3 Programmer’s Model
3.1 Registers ................................................................. 3-2
3.2 Instruction set overview .................................................. 3-6
3.3 Instruction encodings ..................................................... 3-9
3.4 Instruction cycle timing ................................................. 3-16
3.5 Data hazards .............................................................. 3-17
Appendix A  System Connectivity and Signals
A.1  Connecting the MOVE coprocessor ........................................................... A-2
A.2  Signal description ..................................................................................... A-3
List of Tables

MOVE Coprocessor

Change history ............................................................................................................................................... ii
Table 3-1 MOVE register summary ................................................................................................... 3-2
Table 3-2 CR_ACC Register bit assignments ................................................................................. 3-3
Table 3-3 CR_IDX Register bit assignments ................................................................................. 3-3
Table 3-4 CR_BYO Register bit assignments ................................................................................. 3-4
Table 3-5 CR_CFG Register bit assignments ................................................................................. 3-5
Table 3-6 CR_ID Register bit assignments ................................................................................... 3-5
Table 3-7 MOVE instruction classes ............................................................................................ 3-6
Table 3-8 MOVE instruction encoding ......................................................................................... 3-7
Table 3-9 Instruction set summary ............................................................................................... 3-9
Table 3-10 UMCR encoding ............................................................................................................ 3-10
Table 3-11 UMRC encoding ............................................................................................................ 3-10
Table 3-12 UMBBR encoding ......................................................................................................... 3-11
Table 3-13 UMRBB encoding ....................................................................................................... 3-12
Table 3-14 UBBLD encoding ......................................................................................................... 3-13
Table 3-15 USALD encoding ......................................................................................................... 3-14
Table 3-16 Instruction cycle timing ............................................................................................... 3-16
Table 3-17 Data dependencies between instructions ................................................................. 3-17
Table A-1 MOVE instruction fetch interface signals ................................................................. A-3
Table A-2 MOVE data buses ......................................................................................................... A-3
Table A-3 MOVE interface signals ............................................................................................... A-3
| Table A-4 | MOVE miscellaneous signals | A-4 |
List of Figures
MOVE Coprocessor

Key to timing diagram conventions ................................................................. xii
Figure 1-1 MOVE block diagram ........................................................................... 1-3
Figure 2-1 MOVE connections ........................................................................... 2-3
Figure 2-2 MOVE functional diagram ................................................................. 2-4
Figure 3-1 CR_ACC Register bit assignments ....................................................... 3-2
Figure 3-2 CR_IDX Register bit assignments ....................................................... 3-3
Figure 3-3 CR_BYO Register bit assignments ....................................................... 3-4
Figure 3-4 CR_CFG Register bit assignments ....................................................... 3-4
Figure 3-5 CR_ID Register bit assignments .......................................................... 3-5
Preface

This preface introduces the ARM MOVE® (ARM9x6) coprocessor. It contains the following sections:

- *About this manual* on page x
- *Feedback* on page xiv.
About this manual

This is the technical reference manual for the MOVE (ARM9x6) coprocessor that enhances performance of MPEG4 encoder applications. The manual covers the functional specification for both hardware and software. It does not include the implementation details of either the hardware or the software.

Please contact info@arm.com for more information on the MOVE product family and support package.

Intended audience

This manual has been written for hardware and software engineers implementing System-on-Chip designs. It provides information to enable designers to integrate the peripheral into a target system as quickly as possible.

Using this manual

This manual is organized as follows:

Chapter 1 Introduction

Read this chapter for an introduction to the MOVE coprocessor and its features.

Chapter 2 Functional Description

Read this chapter for a description of the major functional blocks of the MOVE coprocessor.

Chapter 3 Programmer’s Model

Read this chapter for a description of the MOVE coprocessor registers and programming details.

Appendix A System Connectivity and Signals

Read this appendix for details of the MOVE coprocessor signals.

Conventions

Conventions that this manual can use are described in:

- Typographical on page xi
- Timing diagrams on page xi
- Signals on page xii
- Numbering on page xiii.
Typographical

The typographical conventions are:

*italic*  
Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

*bold*  
Highlights interface elements, such as menu names. Denotes ARM processor 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* *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 >*  
Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example:

- MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
- The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xii explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.
Key to timing diagram conventions

Signals

The signal conventions are:

**Signal level**
- The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

**Prefix A**
- Denotes AXI global and address channel signals.

**Prefix B**
- Denotes AXI write response channel signals.

**Prefix C**
- Denotes AXI low-power interface signals.

**Prefix H**
- Denotes Advanced High-performance Bus (AHB) signals.

**Prefix n**
- Denotes active-LOW signals except in the case of AXI, AHB, or Advanced Peripheral Bus (APB) reset signals.

**Prefix P**
- Denotes APB signals.

**Prefix R**
- Denotes AXI read data channel signals.

**Prefix W**
- Denotes AXI write data channel signals.

**Suffix n**
- Denotes AXI, AHB, and APB reset signals.
Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of 0x3F. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM Limited periodically provides updates and corrections to its documentation. See http://www.arm.com for current errata sheets, addenda, and the Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the ARM MOVE coprocessor. Refer to the following documents for other relevant information:

- ARM926E-S Engineering Specification (ARM CP023 ESPC 0001).
- ARM946E-S Engineering Specification (ARM CP018 ESPC 0000).
- ARM966E-S Engineering Specification (ARM CP017 ESPC 0000).
Feedback

ARM Limited welcomes feedback on the ARM MOVE coprocessor and its documentation.

Feedback on this ARM MOVE coprocessor

If you have any comments or suggestions about this product, contact your supplier giving:
- the product name
- a concise explanation of your comments.

Feedback on this manual

If you have any comments on this manual, send email to errata@arm.com giving:
- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.
Chapter 1

Introduction

This chapter introduces the ARM MOVE coprocessor. It contains the following sections:

- *About the MOVE coprocessor* on page 1-2
- *MOVE structure* on page 1-3
- *MOVE interface* on page 1-4.
1.1 About the MOVE coprocessor

The MOVE coprocessor is a video encoding acceleration coprocessor designed to accelerate Motion Estimation (ME) algorithms within block-based video encoding schemes such as MPEG4 and H.263. It provides support for the execution of Sum of Absolute Differences (SAD) calculations, which account for most of the processing activity within an ME algorithm. The ME algorithms require many comparisons of 8x8 pixel blocks to be made between a current frame and a reference frame.

This coprocessor is one element in the MOVE product group. It is assigned the letter U. All coprocessor instruction mnemonics are prefixed with the letter U.

——— Note ———————
This document refers to the coprocessor as the MOVE.

——— ——————————
1.2 MOVE structure

The MOVE interprets a set of coprocessor instructions that are part of the ARM instruction set. The ARM coprocessor instructions enable the ARM processor to:

- transfer data from memory to the MOVE, using LDC instructions
- transfer data from MOVE to ARM registers, using MCR instructions, and from ARM to MOVE registers, using MRC instructions.

The ARM processor acts as an address generator and data pump for the MOVE. The MOVE consists of:
- a register bank
- a data path
- control logic.

Figure 1-1 shows the major blocks of the MOVE.

---

Note

For details of the blocks, see Functional overview on page 2-4.
1.3 MOVE interface

The only connection to the MOVE is the coprocessor interface from the ARM processor. All other system connectivity, such as AMBA or interrupts, are handled using the ARM processor.

Note

Connection of multiple external coprocessors is not supported.

For more information on the coprocessor interface, see the applicable ARM9x6 processor Technical Reference Manual.
Chapter 2
Functional Description

This chapter gives a functional description of the ARM MOVE coprocessor, and illustrates its functional and block diagrams. It contains the following sections:

- **MOVE overview** on page 2-2
- **Connectivity** on page 2-3
- **Functional overview** on page 2-4.
2.1 MOVE overview

The MOVE is closely coupled to the ARM9x6 family of processors. Each instruction that is destined for the MOVE is processed immediately as if the ARM core is executing the instruction.

The MOVE is not a fire and forget coprocessor. MOVE instructions are executed immediately and the result is available for the following instruction, subject to certain conditions. See Data hazards on page 3-17. This means that the ARM processor does not have to poll the status of the coprocessor, and the coprocessor does not have to interrupt the ARM processor.
2.2 Connectivity

Figure 2-1 shows the connections of the MOVE to the ARM coprocessor interface.
2.3 Functional overview

Figure 2-2 shows a MOVE functional diagram.

2.3.1 Control logic

The control logic reads each instruction as it arrives from memory. It contains a pipeline follower so that the MOVE and the ARM processor are in step.

The coprocessor pipeline consists of the following stages:
- Fetch
- Decode
- Execute
- Memory
- Write
- SAD accumulate.

2.3.2 Register bank

The register bank holds:
- the block buffer and the MCR/MRC-capable registers
- the logic to increment CR_IDX.

2.3.3 Data path

The data path contains the logic for the USALD instruction, including:
- byte manipulation
- the SAD
- the accumulate.

The data path also handles register read selection and result-forwarding for internal operations, and UMRC and UMRBB instructions.
Chapter 3
Programmer’s Model

This chapter describes the programmer’s model for the ARM MOVE coprocessor. It contains the following sections:

- *Registers* on page 3-2
- *Instruction set overview* on page 3-6
- *Instruction encodings* on page 3-9
- *Instruction cycle timing* on page 3-16
- *Data hazards* on page 3-17.
3.1 Registers

The MOVE coprocessor contains two types of data storage:

**Registers**

These are used to transfer data directly between ARM registers and the coprocessor. These can be accessed by \texttt{MCR} and \texttt{MRC} operations.

**Block buffer**

This an 8-byte (64-bit) by 8-line store. The block buffer is accessed by a set of special MOVE instructions. ARM sees these as \texttt{LDC}, \texttt{MCR}, and \texttt{MRC} instructions.

Table 3-1 shows the MOVE register summary.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Reset value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_ACC</td>
<td>Read/ write</td>
<td>0</td>
<td>See Accumulated Results Register</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>CR_IDX</td>
<td>Read/ write</td>
<td>0</td>
<td>See Line Index Register on page 3-3</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>CR_BYO</td>
<td>Read/ write</td>
<td>0</td>
<td>See Byte Offset Register on page 3-3</td>
</tr>
<tr>
<td>CR_CFG</td>
<td>Read/ write</td>
<td>0</td>
<td>See Configuration Register on page 3-4</td>
</tr>
<tr>
<td>CR_ID</td>
<td>Read</td>
<td>41001020</td>
<td>See Identification Register on page 3-5</td>
</tr>
</tbody>
</table>

3.1.1 Accumulated Results Register

The CR_ACC Register is a 14-bit read/write register. You can update it directly using \texttt{MCR} instructions and indirectly using SAD operations. Figure 3-1 shows the CR_ACC Register bit assignments.

![Figure 3-1 CR_ACC Register bit assignments](image-url)
Table 3-2 lists the CR_ACC Register bit assignments.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:14]</td>
<td>-</td>
<td>Reserved, return to zero on reads, writes ignored.</td>
</tr>
<tr>
<td>[13:0]</td>
<td>ACC</td>
<td>Stores the accumulated result.</td>
</tr>
</tbody>
</table>

### 3.1.2 Line Index Register

The CR_IDX Register is a 3-bit read/write register. You can update it directly using \texttt{MCR} instructions. You can update it indirectly using block buffer loads or stores, or SAD operations that increment the line index. Figure 3-2 shows the CR_IDX Register bit assignments.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Reserved, masked on reads, set to zero for writes.</td>
</tr>
<tr>
<td>[2:0]</td>
<td>IDX</td>
<td>Indicates the line index for the block buffer. This sets which line is referenced by the next operation that accesses the block buffer. These operations only ever use a single line from the block buffer. When this register is incremented past the value 7, it wraps to zero.</td>
</tr>
</tbody>
</table>

### 3.1.3 Byte Offset Register

The CR_BYO Register is a 2-bit read/write register. You can only update it using \texttt{MCR} instructions. Figure 3-3 on page 3-4 shows the CR_BYO Register bit assignments.
3.1.4 Configuration Register

The CR_CFG Register is a single-bit read/write register. You can only update it using MCR instructions. shows the CR_CFG Register bit assignments.

Table 3-4 CR_CFG Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:2]</td>
<td>-</td>
<td>Reserved, return to zero on reads, writes ignored.</td>
</tr>
<tr>
<td>[1:0]</td>
<td>BYO</td>
<td>The MOVE supports accesses to the reference frame from byte-aligned addresses. The lower two bits of the address bus for these loads are stored in this register.</td>
</tr>
</tbody>
</table>

**Note**

The coprocessor has no direct visibility of the address value used by the ARM processor for memory accesses, so software must program the byte offset into CR_CFG separately. Also, the ARM processor does not directly support nonword-aligned loads for coprocessors, so the MOVE performs three word loads and then extracts the required eight bytes.
Table 3-5 lists the CR_CFG Register bit assignments.

### Table 3-5 CR_CFG Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:1]</td>
<td>-</td>
<td>Reserved, masked on reads, set to zero for writes.</td>
</tr>
<tr>
<td>[0]</td>
<td>IDX_INC</td>
<td>This bit controls whether the CR_IDX register is incremented after a block buffer load/store or SAD operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = no increment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = CR_IDX is incremented after the block buffer or SAD operation completes</td>
</tr>
</tbody>
</table>

#### 3.1.5 Identification Register

The CR_ID Register is a 32-bit read-only register that contains the MOVE architecture and revision code. shows the CR_ID Register bit assignments.

![Figure 3-5 CR_ID Register bit assignments](image)

Table 3-6 lists the CR_ID Register bit assignments.

### Table 3-6 CR_ID Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:24]</td>
<td>Implementor. The code number for the implementor. 0x41 = ARM.</td>
</tr>
<tr>
<td>[15:8]</td>
<td>Part number of the MOVE. 0x10 = MOVE coprocessor.</td>
</tr>
<tr>
<td>[7:4]</td>
<td>Contains the major version number for the implementation.</td>
</tr>
<tr>
<td>[3:0]</td>
<td>Contains the minor version number for the implementation.</td>
</tr>
</tbody>
</table>
3.2 Instruction set overview

The assembler syntax of the MOVE coprocessor uses the same format as that of the ARM processor, where:

{} Indicates an optional field.

cond Is the ARM instruction condition code field.

dest Specifies the MOVE destination register.

Rn Is an ARM register.

CRn Is a MOVE register.

CBBn Is a word in the MOVE block buffer.

! Indicates that the calculated address must be written back to the base register.

UCP Is MOVE coprocessor number.

8_bit_offset Is an expression evaluating to an 8-bit word offset. This offset is added to the base register to form the load address. The offset must be a multiple of four.

Note Post indexing with or without write-back is allowed.

3.2.1 MOVE coprocessor instruction classes

The MOVE instructions are in the coprocessor instruction classes shown in Table 3-7.

<table>
<thead>
<tr>
<th>MOVE instruction</th>
<th>ARM coprocessor encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCR/UMBRR</td>
<td>MCR</td>
</tr>
<tr>
<td>UMRC/UMRBB</td>
<td>MRC</td>
</tr>
<tr>
<td>UBBLD</td>
<td>LDC (multiple load)</td>
</tr>
<tr>
<td>USALD</td>
<td>LDC (multiple load)</td>
</tr>
</tbody>
</table>
The MOVE instructions are encoded as shown in Table 3-8.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>cond</th>
<th>UCP</th>
<th>Rn</th>
<th>UCP</th>
<th>CRd</th>
<th>CRn</th>
<th>CBBd</th>
<th>Rn</th>
<th>UCP</th>
<th>CBBd</th>
<th>Rd</th>
<th>UCP</th>
<th>8_bit_offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCR</td>
<td>1 1 1 0 0 0 0 0 0</td>
<td>0 0 0 1 0 0 0 0</td>
<td>CRd</td>
<td>Rn</td>
<td>UCP</td>
<td>0 0 0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMRC</td>
<td>1 1 1 0 0 0 0 0 1</td>
<td>0 0 0 1 0 0 0 0</td>
<td>CRn</td>
<td>Rd</td>
<td>UCP</td>
<td>0 0 0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMBBR</td>
<td>1 1 1 0 0 0 0 0 0</td>
<td>0 0 1 1 0 0 0 0 0</td>
<td>CBBd</td>
<td>Rn</td>
<td>UCP</td>
<td>0 0 1 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMRBB</td>
<td>1 1 1 0 0 0 0 0 1</td>
<td>0 0 1 1 0 0 0 0 0</td>
<td>CBBn</td>
<td>Rd</td>
<td>UCP</td>
<td>0 0 1 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UBLBD</td>
<td>1 1 0 0 P U 1 W 1</td>
<td>1 0 0 0 0 0</td>
<td>Rn</td>
<td>1 0 0 0</td>
<td>UCP</td>
<td>8_bit_offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USALD</td>
<td>1 1 0 0 P U 1 W 1</td>
<td>1 0 0 0 0</td>
<td>Rn</td>
<td>1 0 0 0</td>
<td>UCP</td>
<td>8_bit_offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- Note ---

Only the following opcodes are valid for the MOVE. Any variations on these opcodes (which retain UCP in bits 11 down to 8) results in unpredictable behavior.

- **cond** is the condition code.
- **UCP** is the MOVE coprocessor number.
- **Rn** is the ARM register source.
- **CRn** is the MOVE register source.
- **CBBn** is the block buffer source word.
- **Rd** is the ARM register destination.
- **CRd** is the MOVE register destination.
- **CBBd** is the block buffer destination word.
- **8_bit_offset** is an 8-bit number (0-255) that is used to indicate an address offset.
- **P** is the pre- or post-indexing bit:
  - 0 = Post-indexing. Add offset after transfer.
  - 1 = Pre-indexing. Add offset before transfer.
- **U** is the up or down bit:
  - 0 = Down. Subtract from base.
  - 1 = Up. Add offset to base.
W     Is the writeback bit.
0 = No writeback.
1 = Write address into base.
3.3 Instruction encodings

This section details the instruction encodings. Table 3-9 shows the MOVE instruction set summary and gives the page reference for the detailed description of each instruction.

<table>
<thead>
<tr>
<th>MOVE instruction</th>
<th>Description</th>
<th>Detailed description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCR</td>
<td>Moves to a MOVE register from an ARM register</td>
<td>UMCR on page 3-10</td>
</tr>
<tr>
<td>UMRC</td>
<td>Moves to an ARM register from a MOVE register</td>
<td>UMRC on page 3-10</td>
</tr>
<tr>
<td>UMBBR</td>
<td>Moves to a word in the MOVE block buffer from an ARM register (for debug only)</td>
<td>UMBBR on page 3-11</td>
</tr>
<tr>
<td>UMRBB</td>
<td>Moves to an ARM register from a word in the MOVE block buffer (for debug only)</td>
<td>UMRBB on page 3-12</td>
</tr>
<tr>
<td>UBBLD</td>
<td>Loads two words from memory into the block buffer</td>
<td>UBBLD on page 3-13</td>
</tr>
<tr>
<td>USALD</td>
<td>Performs a byte-aligned load of 8-bytes from memory, performs SAD and accumulates in the CR_ACC Register</td>
<td>USALD on page 3-14</td>
</tr>
</tbody>
</table>
3.3.1 UMCR

The UMCR instruction writes to the MOVE registers. UMCR moves data from ARM register Rn to MOVE register CRd. Table 3-10 shows the UMCR encoding.

**Syntax**

UMCR CRd, Rn, {cond}

**Example**

UMCR CR_ACC, R0 ; load contents of R0 into CR_ACC

3.3.2 UMRC

The UMRC instruction reads from the MOVE registers. UMRC moves data from a MOVE register CRn to an ARM register Rd. Table 3-11 shows the UMRC encoding.

**Syntax**

UMRC Rd, CRn, {cond}

**Examples**

UMRC R6, CR_ID ; load ID register into R6.
The UMBBR instruction writes to a word in the MOVE block buffer. UMBBR moves data from an ARM register Rn to CBBd in the MOVE block buffer. Table 3-12 shows the UMBBR encoding.

### Syntax

UMBBR CBBd, Rn, {cond}

### Example

UMBBR 3, R0

; load contents of R0 into the second word
; of the second line of the block buffer.

### Notes

- This instruction is intended only for use by the MultiICE debugger.
- For the UMBBR instruction, the CBBd opcode field represents a look up into the block buffer:
  - **Bits[19:17]** Refers to the block buffer line number.
  - **Bit[16]** Is 0 for the first word from memory (the first word loaded by a UBBLD operation), and 1 for the second word.
3.3.4 UMRBB

The UMRBB instruction reads a word from the MOVE block buffer. UMRBB moves a word of data from the MOVE block buffer to an ARM register Rd. Table 3-13 shows the UMRBB encoding.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>UMRBB Rd, CBBn, {cond}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>UMRBB R6, 4</td>
</tr>
<tr>
<td></td>
<td>; load first word from the third line</td>
</tr>
<tr>
<td></td>
<td>; of the block buffer into R6.</td>
</tr>
</tbody>
</table>

Notes

- This instruction is intended only for use by the MultiICE debugger.
- For the UMRBB instruction, the CBBn opcode field represents a look up into the block buffer:
  - **Bits[19:17]** Refers to the block buffer line number.
  - **Bit[16]** Is 0 for the first word from memory (the first word loaded by a UBBLD operation), and 1 for the second word.
3.3.5 UBBLD

The UBBLD instruction loads data into the block buffer. Table 3-14 shows the UBBLD encoding.

Table 3-14 UBBLD encoding

<table>
<thead>
<tr>
<th>cond</th>
<th>P</th>
<th>U</th>
<th>W</th>
<th>Rn</th>
<th>UCP</th>
<th>8_bit_offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3 2 1 0 9</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1 8 7 6 5 4 3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8_bit_offset</td>
</tr>
</tbody>
</table>

Syntax

UBBLD [Rn], #0, {cond}
UBBLD [Rn, #/+/-10_Bit_Offset]!, {cond}
UBBLD [Rn], #/+/-10_Bit_Offset!, {cond}

Operation

Loadword Block_Buffer(CR_IDX,0)
Loadword Block_Buffer(CR_IDX,1)
If (IDX_INC==1)
++CR_IDX

Example

UBBLD [R0], #320!
; Load two words into block buffer from mem(R0)
; and post-increment R0 to next line
3.3.6 USALD

The USALD instruction loads data from the reference block and performs a SAD accumulate operation. Table 3-15 shows the USALD encoding.

Table 3-15 USALD encoding

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>cond</td>
<td>P</td>
<td>U</td>
<td>W</td>
<td>Rn</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>UCP</td>
<td>8_bit_offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Syntax

USALD  [Rn], #0, {cond}
USALD  [Rn, +/-10_Bit_Offset]{!}, {cond}
USALD  [Rn], #+/-10_Bit_Offset{!}, {cond}

Operation

If (CR_BYO==0)
Load word to firstword_tmp
else
Load word to mux1_tmp;
Load word to mux2_tmp;
Based on CR_BYO combine mux1_tmp and mux2_tmp to form non-aligned word load in firstword_tmp.

For each byte in firstword_tmp and each corresponding byte in Block_Buffer(CR_IDX, 0) perform SAD. Accumulate results in CR_ACC.

If (CR_BYO==0)
Load word to secondword_tmp
else
Load word to mux3_tmp;
Based on CR_BYO combine mux2_tmp and mux3_tmp to form non aligned word load in secondword_tmp.

For each byte in secondword_tmp and each corresponding byte in Block_Buffer(CR_IDX, 1) perform SAD. Accumulate results in CR_ACC.

If (IDX_INC==1)
++CR_IDX

Example

USALD  [R0], #320!  ; Process two words from mem(R0), perform SAD
                    ; and post-increment R0 to next line.
Notes

Arithmetic is unsigned. No facilities are included to detect or handle the overflow of CR_ACC. The size of CR_ACC has been chosen to ensure that overflow cannot occur from an 8x8 block compare.
3.4 Instruction cycle timing

Table 3-16 shows the number of cycles each instruction takes to complete.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCR</td>
<td>1</td>
</tr>
<tr>
<td>UMRC</td>
<td>1</td>
</tr>
<tr>
<td>UMBBR</td>
<td>1</td>
</tr>
<tr>
<td>UMRBB</td>
<td>1</td>
</tr>
<tr>
<td>UBBLD</td>
<td>2</td>
</tr>
<tr>
<td>USALD</td>
<td>N cycles</td>
</tr>
</tbody>
</table>

**Note**

N = 2 cycles if CR_BYO is zero, otherwise N = 3 cycles.
3.5 Data hazards

Data hazards are circumstances where data dependencies between instructions can result in unpredictable behavior.

There are no hardware interlocks in the MOVE to cope with data hazards. Instead, the software must include a noncoprocessor instruction where necessary. This can be a NOP.

The dependencies are shown in Table 3-17.

### Table 3-17 Data dependencies between instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Next instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCR</td>
<td>1. No UMRC which uses CRd for 2 cycles</td>
</tr>
<tr>
<td></td>
<td>2. If CRd is CR_Byo then no USALD for 2 cycles</td>
</tr>
<tr>
<td>UMRC</td>
<td>1. Any.</td>
</tr>
<tr>
<td>UBBLD</td>
<td>when INC_IDX is clear 1. No UMRBB for 1 cycle</td>
</tr>
<tr>
<td></td>
<td>when INC_IDX is set 1. No UMRC CR_IDX for 2 cycles</td>
</tr>
<tr>
<td></td>
<td>2. No UMRBB for 1 cycle</td>
</tr>
<tr>
<td>USALD</td>
<td>when INC_IDX is clear 1. No UMCR CR_ACC for 1 cycle</td>
</tr>
<tr>
<td></td>
<td>2. No UMCR CR_ACC for 2 cycles</td>
</tr>
<tr>
<td></td>
<td>3. No UMRBB for 1 cycle</td>
</tr>
<tr>
<td></td>
<td>when INC_IDX is set 1. No UMCR CR_ACC for 1 cycle</td>
</tr>
<tr>
<td></td>
<td>2. No UMCR CR_ACC for 2 cycles</td>
</tr>
<tr>
<td></td>
<td>3. No UMCR CR_IDX for 2 cycles</td>
</tr>
<tr>
<td></td>
<td>4. No UMRBB for 1 cycle</td>
</tr>
</tbody>
</table>
Appendix A
System Connectivity and Signals

This appendix describes the connectivity and signals for the ARM MOVE coprocessor. It contains the following sections:

- Connecting the MOVE coprocessor on page A-2
- Signal description on page A-3.
A.1 Connecting the MOVE coprocessor

The MOVE is designed to be connected directly to ARM9x6 processors. The CHSDE_C and CHSEX_C outputs are unused in this configuration. For the ARM966 and the ARM946 processors the CPABORT input must be connected to the ETMDABORT pin and the ETM must be enabled. For the ARM926 processor, the CPABORT pin must be connected to the CPABORT pin of the processor, and the CPTBIT pin to the nCPINSTRVALID pin of the processor.
A.2 Signal description

Table A-1 to Table A-4 on page A-4 show the signals for the MOVE to interface to the ARM9x6.

### Table A-1 MOVE instruction fetch interface signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPINSTR[31:0]</td>
<td>Input</td>
<td>Coprocessor instruction data. This is the coprocessor instruction data bus over which instructions are transferred to the pipeline follower in the coprocessor.</td>
</tr>
<tr>
<td>CPTBIT</td>
<td>Input</td>
<td>Coprocessor Thumb bit. If HIGH, the coprocessor interface is in the Thumb state.</td>
</tr>
<tr>
<td>nCPMREQ</td>
<td>Input</td>
<td>Not coprocessor memory request. When LOW on a rising CPCLK edge and CPCLKEN HIGH, the instruction on CPINSTR must enter the decode stage of the coprocessor pipeline follower. The second instruction previously in the pipeline followers decode stage enters its execute stage.</td>
</tr>
</tbody>
</table>

### Table A-2 MOVE data buses

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPDIN[31:0]</td>
<td>Output</td>
<td>Data into the ARM. The coprocessor data bus for transferring MRC data from the coprocessor to the ARM.</td>
</tr>
<tr>
<td>CPDOU[31:0]</td>
<td>Input</td>
<td>Data out from ARM. The coprocessor data bus for transferring MCR and LDC data to the coprocessor.</td>
</tr>
</tbody>
</table>

### Table A-3 MOVE interface signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHSDE[1:0]</td>
<td>Output</td>
<td>Coprocessor handshake decode. The handshake signals from the decode stage of the coprocessor pipeline follower.</td>
</tr>
<tr>
<td>CHSDE_C[1:0]</td>
<td>Output</td>
<td>A combinatorial version of CHSDE, output a cycle in advance of CHSDE (but only valid towards the end of the cycle). Only for use with a 920 retiming wrapper.</td>
</tr>
<tr>
<td>CHSEX[1:0]</td>
<td>Output</td>
<td>Coprocessor handshake execute. The handshake signals from the execute stage of the coprocessor pipeline follower.</td>
</tr>
<tr>
<td>CHSEX_C[1:0]</td>
<td>Output</td>
<td>A combinatorial version of CHSEX, output a cycle in advance of CHSEX (but only valid towards the end of the cycle). Only for use with a 920 retiming wrapper.</td>
</tr>
</tbody>
</table>
### Table A-3 MOVE interface signals (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPABORT</td>
<td>Input</td>
<td>Coprocessor operation data abort. This signal is used by the coprocessor to recover from an abort during the data loads.</td>
</tr>
<tr>
<td>CPLATECANCEL</td>
<td>Input</td>
<td>Coprocessor late cancel. When a coprocessor instruction is being executed, if this signal is HIGH during the first memory cycle, the coprocessor instruction must be canceled without having updated the coprocessor state.</td>
</tr>
<tr>
<td>CPPASS</td>
<td>Input</td>
<td>Coprocessor pass. This signal indicates that there is a coprocessor instruction in the execute stage of the pipeline, and it must be executed.</td>
</tr>
</tbody>
</table>

### Table A-4 MOVE miscellaneous signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGEND</td>
<td>Input</td>
<td>Static configuration signal. HIGH to support big-endian systems and LOW for little-endian. Usually supplied from the processor (for example, BIGENDOUT from 920).</td>
</tr>
<tr>
<td>CPCLK</td>
<td>Input</td>
<td>Clock. This clock times all coprocessor memory accesses (both data and instruction) and internal operations.</td>
</tr>
<tr>
<td>CPCLKEN</td>
<td>Input</td>
<td>Coprocessor clock enable. The coprocessor clock CPCLK is qualified by this signal.</td>
</tr>
<tr>
<td>nRESET</td>
<td>Input</td>
<td>Not reset. This is a level-sensitive input signal, which is used to reset the coprocessor. The coprocessor asynchronously reset when nRESET goes LOW.</td>
</tr>
<tr>
<td>SCANCLK</td>
<td>Input</td>
<td>Clock input for production scan testing.</td>
</tr>
<tr>
<td>SCANIN</td>
<td>Input</td>
<td>Test signal.</td>
</tr>
<tr>
<td>SCANMODE</td>
<td>Input</td>
<td>Test mode control input for production scan testing.</td>
</tr>
<tr>
<td>SCANOUT</td>
<td>Output</td>
<td>Test signal.</td>
</tr>
</tbody>
</table>
Index

A
Architecture version 3-5
ARM9x6 A-2
ARM946 A-2
ARM966 A-2

B
Block buffer 2-4, 3-2
loading data into 3-13
reading from 3-12
writing to 3-11

C
Control logic 2-4
Conventions numerical xiii
signal naming xii
timing diagram xi

typographical xi
Coprocessor instruction classes 3-6
Coprocessor interface 1-4
Coprocessor interface signals A-3
CR_ACC register 3-2
CR_BYO register 3-3
CR_CFG register 3-4
CR_ID register 3-5
CR_IDX register 3-3

D
Data buses signals A-3
Data dependencies 3-17
Data hazards 3-17
Data path 2-4
Data storage 3-2

H
Hardware interlocks 3-17

I
Instruction cycle timing 3-16
Instruction fetch interface signals A-3
Instructions dependencies between 3-17
Interface signals A-3

L
Loading data from the reference block 3-14
Loading data into the block buffer 3-13

M
Miscellaneous signals A-4
Mobile Optimised Video Extensions.
see MOVE.
MOVE 1-3
miscellaneous signals A-4
Index

revision code 3-5
structure 1-3
MOVE coprocessor
   block diagram 2-4
   connectivity 2-3, A-2
   function description 2-2
   function diagram 2-3
MOVE instruction set
   encoding 3-7, 3-9
   instruction classes 3-6
   overview 3-6
   UMBBR 3-11
   UMBLD 3-13
   UMCR 3-10
   UMRBB 3-12
   UMRC 3-10
   USALD 3-14

S
   SAD accumulate operation 3-14
   Signal naming conventions xii
   Signals A-3
      data buses A-3
      instruction fetch interface A-3
      interface A-3
      miscellaneous A-4

T
   Timing diagram conventions xi
   Typographical conventions xi

U
   UMBBR instruction 3-11
   UMBLD instruction 3-13
   UMCR instruction 3-10
   UMRBB instruction 3-12
   UMRC instruction 3-10
   USALD instruction 3-14

W
   Writing to MOVE registers 3-10
   Writing to the block buffer 3-11

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

N
   P
   Pipeline stages 2-4

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii

P
   Pipeline stages 2-4

R
   Reading from MOVE registers 3-10
   Reading from the block buffer 3-12
   Reference block
      loading data from 3-14
   Register bank 2-4
   Registers 3-2
      CR_ACC 3-2
      CR_BYO 3-3
      CR_CFG 3-4
      CR_ID 3-5
      CR_IDX 3-3
      reading from 3-10
      summary of 3-2
      writing to 3-10
   Numerical conventions xiii