# ARM968E-S™

## Revision: r0p1

# Technical Reference Manual

ARM®

Copyright © 2004, 2006 ARM Limited. All rights reserved.

ARM DDI 0311D

# ARM968E-S
## Technical Reference Manual

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Release Information

**Change history**

| Date | Issue | Confidentiality | Change |
|---|---|---|---|
| 30 January 2004 | A | Non-Confidential | First r0p0 release |
| 10 March 2004 | B | Non-Confidential | Second release with TCM access clarification |
| 04 November 2004 | C | Non-Confidential | Third release with corrections of address bus widths in Figures 6-11, 6-12, and 6-13 |
| 08 November 2006 | D | Non-Confidential | Updated for r0p1. Maintenance release. |

### Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited 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 Limited 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.

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
# ARM968E-S Technical Reference Manual

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## Chapter 4 System Control Coprocessor

## Chapter 5 Bus Interface Unit

## Chapter 6 Tightly-Coupled Memory Interface

## Chapter 7 DMA Interface

## Chapter 8 Debug Support

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## Chapter 9 Embedded Trace Macrocell Interface

## Chapter 10 Test Support

## Chapter 11 DFT Interface

## Appendix A Signal Descriptions

## Appendix B AC Parameters

## Glossary

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

# List of Tables
# ARM968E-S Technical Reference Manual

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# List of Figures
# ARM968E-S Technical Reference Manual

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Preface

This preface introduces the *ARM968E-S Technical Reference Manual*. It contains the following sections:

- *About this manual* on page xii
- *Feedback* on page xvi.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## About this manual

This manual is the *Technical Reference Manual* (TRM) for the ARM968E-S processor.

### Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this manual, 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 manual is written to help designers, system integrators, and verification engineers who are implementing systems around the ARM968E-S processor.

### Using this manual

This manual is organized into the following chapters:

**Chapter 1 *Introduction***

Read this chapter for an introduction to the ARM968E-S processor.

**Chapter 2 *Programmer's Model***

Read this chapter for an introduction to the ARM programmer's model.

**Chapter 3 *Memory Map***

Read this chapter for a description of the ARM968E-S fixed memory map implementation.

**Chapter 4 *System Control Coprocessor***

Read this chapter for a description of the ARM968E-S CP15 control and status registers.

**Chapter 5 *Bus Interface Unit***

Read this chapter for a description of the operation of the *Bus Interface Unit* (BIU) and the AHB write buffer.

**Chapter 6 *Tightly-Coupled Memory Interface***

Read this chapter for a description of the requirements and operation of the tightly-coupled memory.

**Chapter 7** ***DMA Interface***

Read this chapter for a description of the ARM968E-S *Direct Memory Access* (DMA) interface.

**Chapter 8** ***Debug Support***

Read this chapter for a description of the debug support for the ARM968E-S processor and the EmbeddedICE-RT logic.

**Chapter 9** ***Embedded Trace Macrocell Interface***

Read this chapter for a description of the ETM interface, including descriptions of how to enable the interface.

**Chapter 10** ***Test Support***

Read this chapter for a description of the test methodology used for the ARM968E-S synthesized logic and tightly-coupled memory.

**Chapter 11** ***DFT Interface***

Read this chapter for a description of the ARM968E-S Design For Test (DFT) interface.

**Appendix A** ***Signal Descriptions***

Read this appendix for a description of the ARM968E-S signals.

**Appendix B** ***AC Parameters***

Read this appendix for the definitions of the ARM968E-S processor timing parameters.

**Glossary** Read the Glossary for definitions of terms used in this manual.

## Conventions

This section describes the documentation conventions used in this manual:

- *Typographical*
- *Timing diagrams* on page xiv
- *Signals* on page xv.

### Typographical

The typographical conventions are:

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

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**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` 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.

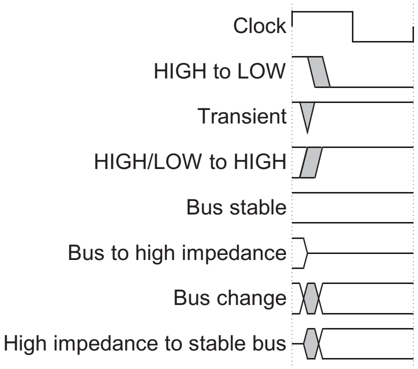
**`monospace bold`** Denotes language keywords when used outside example code.

**< and >** Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font without brackets 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* explains the symbols 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.



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

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

**Prefix P** Denotes APB signals.

**Suffix n** AHB **HRESETn** and APB **PRESETn** reset signals.

## 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 ARM Limited Frequently Asked Questions list.

### ARM publications

This manual contains information that is specific to the ARM968E-S processor. See the following documents for other related information:

- *ARM Architecture Reference Manual* (ARM DDI 0100)
- *ARM9E-S Technical Reference Manual* (ARM DDI 0240)
- *AMBA Specification* (ARM IHI 0011)
- *ARM968E-S Implementation Guide* (ARM DII 0090)
- *AHB Example AMBA System Technical Reference Manual* (ARM DDI 0170)
- *ETM9 Technical Reference Manual* (ARM DDI 0157)
- *ETM9 Implementation Guide* (ARM DII 0001).

### Other publications

This section lists relevant documents published by third parties:

- IEEE Std. 1149.1- 1990, *Standard Test Access Port and Boundary-Scan Architecture*.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Feedback

ARM Limited welcomes feedback on the ARM968E-S processor and its documentation.

## Feedback on the ARM968E-S processor

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.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

# Chapter 1
# Introduction

This chapter introduces the ARM968E-S processor. It contains the following sections:

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 1.1 About the ARM968E-S processor

The synthesizable ARM968E-S processor is a member of the ARM9 Thumb family and implements the ARMv5TE architecture. It supports the 32-bit ARM instruction set and the 16-bit Thumb instruction set. The ARM968E-S processor is targeted at a wide range of embedded applications that require high performance, low system cost, small die size, and low power.

For a description of the ARM and Thumb instruction sets, see the *ARM Architecture Reference Manual*.

Features of the ARM968E-S processor include:

- the ARM9E-S integer core
- *Instruction Tightly-Coupled Memory* (ITCM) and *Data Tightly-Coupled Memory* (DTCM) interfaces with:
  - configurable to sizes of 0KB and 1KB-4MB in power-of-two increments
  - ITCM and DTCM write buffers.
- AHB-Lite *Direct Memory Access* (DMA) interface
- AHB-Lite *Bus Interface Unit* (BIU) interface
- fixed memory map
- optional ETM interface
- optional full debug or reduced debug interface
- scan test support.

Figure 1-1 on page 1-3 shows the blocks of the ARM968E-S processor and their interface signals.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

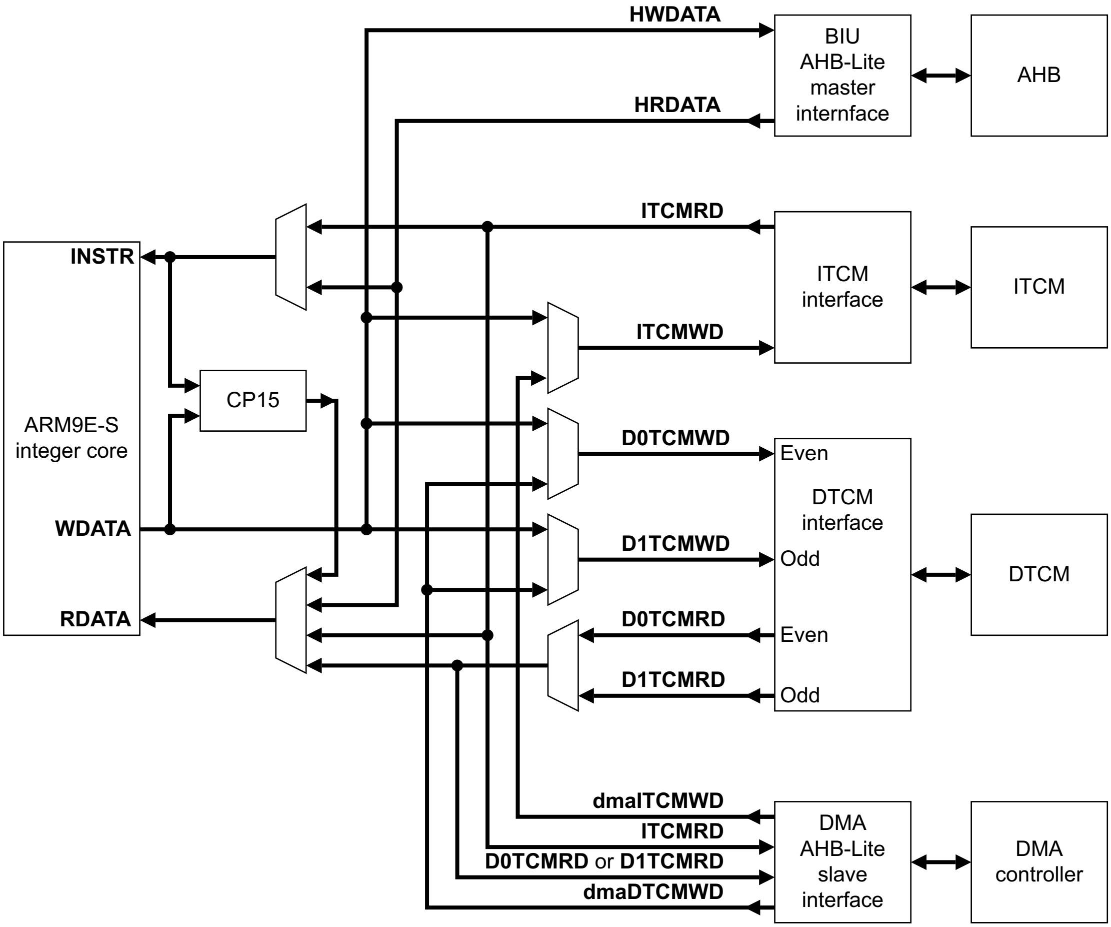


**Figure 1-1 ARM968E-S processor block diagram**

Table 1-1 shows the chapters that describe the blocks in Figure 1-1.

**Table 1-1 Location of block descriptions**

| Block | Location of description |
|---|---|
| ARM9E-S core | *ARM9E-S (Rev 1) Technical Reference Manual* |
| ITCM and DTCM interfaces | Chapter 6 *Tightly-Coupled Memory Interface* |
| BIU AHB-Lite master interface | Chapter 5 *Bus Interface Unit* |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Table 1-1 Location of block descriptions (continued)

| Block | Location of description |
|---|---|
| DMA AHB-Lite slave interface | Chapter 7 *DMA Interface* |
| ETM interface | Chapter 9 *Embedded Trace Macrocell Interface* |
| Debug interface | Chapter 8 *Debug Support* |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 1.2 TCM access

The ARM968E-S processor contains a BIU AHB-Lite master interface and a DMA AHB-Lite slave interface.

### 1.2.1 BIU AHB-Lite master interface

The BIU interface is an interface to AMBA system memory.

### 1.2.2 DMA AHB-Lite slave interface

The DMA interface has priority access to the TCM. It enables an external DMA controller to move real-time data blocks directly into the TCM for more processing without stalling the processor.

The DMA interface accesses the DTCM through two separate ports, D0TCM and D1TCM. The processor and the DMA alternately access the D0TCM and D1TCM ports on a word boundary basis. This unique feature enables the DMA port to move external data blocks into the DTCM without stalling processor access during the DMA block move. Using an even-odd-even-odd word-addressing scheme, the DMA can fill the DTCM while the processor interleaves its addresses for simultaneous full-speed access. Interleaving processor and DMA access to the DTCM gives a unique system-level advantage for real-time data processing applications.

The DMA interface accesses the ITCM through a single port. This enables the DMA to load power-up data directly into the ITCM from a slower memory device such as a flash memory. The processor can then boot from the much faster ITCM array. This is also a convenient mechanism to download new literal tables directly into the ITCM for programmable customer product upgrades.

The DMA controller cannot access the AHB bus. DMA accesses go only to the TCMs. The DMA controller can access the TCMs when the ARM968E-S processor is not in RUN mode, even if the processor has not enabled the TCM interfaces.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 1.3 Debug interface configurations

You can synthesize the ARM968E-S processor with the reduced debug option or the full debug option. See the *ARM968E-S Implementation Guide* for specific synthesis instructions for configuring the debug interface.

### 1.3.1 Reduced debug interface

The default configuration is a minimum-gate-count implementation that does not contain the ETM interface and breakpoint registers normally found on ARM processors. The only debug interface in this configuration is through the IEEE-1149 JTAG port.

### 1.3.2 Full debug interface

The second configuration is the higher-gate-count full debug implementation that contains the ETM interface and watchpoint and breakpoint registers that are useful in software debug environments.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 2
# Programmer's Model

This chapter describes the ARM968E-S registers and provides information for programming the microprocessor. It contains the following sections:

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 2.1 About the programmer's model

The ARM968E-S processor implements ARMv5TE architecture, which includes the 32-bit ARM instruction set and the 16-bit Thumb instruction set. For descriptions of both the ARM and Thumb instruction sets, see the *ARM Architecture Reference Manual*.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 2.2 Processor states

The ARM968E-S processor has two operating states:

**ARM state** 32-bit, word-aligned ARM instructions are executed in this state.

**Thumb state** 16-bit, halfword-aligned Thumb instructions.

In Thumb state, the *Program Counter* (PC) uses bit 1 to select between alternate halfwords.

**Note**

Transition between ARM and Thumb states does not affect the processor mode or the register contents.

### 2.2.1 Switching state

You can switch the processor between ARM state and Thumb state by:

- using the BX and BLX instructions
- loading the PC with the *Load Thumb* (LT) bit cleared in the CP15 c1 Control Register.

The processor begins all exception handling in ARM state. If an exception occurs in Thumb state, the processor changes to ARM state. The change back to Thumb state occurs automatically on return from exception handling.

### 2.2.2 Switching state during exception handling

An exception handler can put the processor in Thumb state, but it must return to ARM state to enable the exception handler to terminate correctly.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 2.3 Processor operating modes

There are seven processor modes of operation:

**User** The nonprivileged mode for normal program execution.

**Fast interrupt (FIQ)**

The privileged exception mode for handling fast interrupts.

**Interrupt (IRQ)**

The privileged exception mode for handling regular interrupts.

**Supervisor** The privileged mode for operating system functions.

**Abort** The privileged exception mode for handling Data Aborts and Prefetch Aborts.

**System** The privileged user mode for operating system functions.

**Undefined** The privileged exception mode for handling Undefined instructions.

Modes other than User mode are collectively known as privileged modes. Privileged modes are used to service exceptions or to access protected resources.

## 2.4 Registers

The ARM968E-S processor has 37 32-bit registers:

- 16 general-purpose registers
- 1 Current Program Status Register
- 15 banked (alternate), mode-specific, general-purpose registers
- 5 banked (alternate), mode-specific Saved Program Status Registers.

These registers are not all accessible at the same time. The processor state and processor operating mode determine which registers are available to the programmer.

Figure 2-1 shows the ARM968E-S register set.

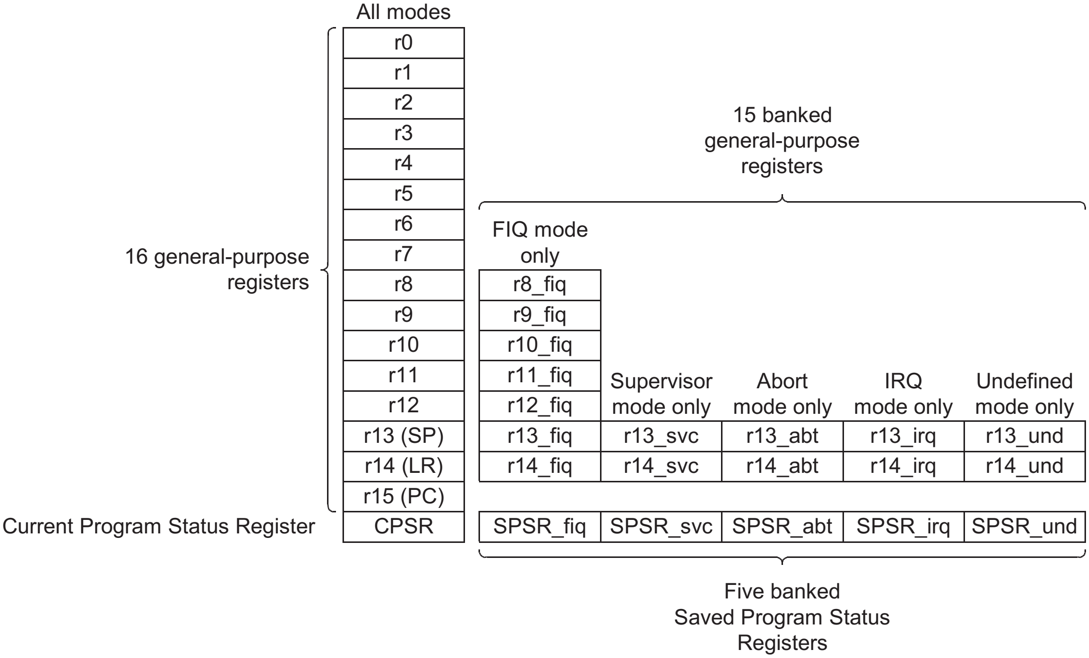


**Figure 2-1 ARM968E-S register set**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

The banked registers are discrete physical registers in the processor that are mapped to the available registers depending on the current processor operating mode. The contents of banked registers are preserved across operating mode changes. Each banked register has a mode identifier to indicate the operating mode. Table 2-1 lists the banked register mode identifiers.

**Table 2-1 Banked register mode identifiers**

| Mode | Identifier |
|---|---|
| User[a] | _usr[b] |
| Fast interrupt | _fiq |
| Interrupt | _irq |
| Supervisor | _svc |
| Abort | _abt |
| System | _usr |
| Undefined | _und |

[a] User mode and System mode use the same registers.

[b] The _usr identifier is omitted unless it is necessary to distinguish the User or System mode register from another banked register.

The banked r13 and r14 general-purpose registers can be used as mode-specific stack pointers and link registers. For fast interrupt handling, the seven banked general-purpose FIQ mode registers, r8_fiq-r14, can be used to reduce the overhead of saving registers.

The r13, r14, and r15 general-purpose registers also have the following special functions:

**Stack pointer** By convention, r13 is used as the *Stack Pointer* (SP).

**Link register** Register r14 is the subroutine *Link Register* (LR).

The LR receives the return address from r15 when a *Branch with Link* (BL or BLX) instruction is executed.

At all other times, you can treat r14 as a general-purpose register The banked r14 registers r14_svc, r14_irq, r14_fiq, r14_abt, and r14_und are similarly used to hold the return values when exceptions arise, or when BL or BLX instructions are executed within interrupt or exception routines.

**Program counter** Register r15 is the *Program Counter* (PC).

In ARM state, bits [1:0] of r15 are Undefined and must be ignored. Bits [31:2] contain the program counter value.

In Thumb state, bit 0 is Undefined and must be ignored. Bits [31:1] contain the program counter value.

### 2.4.1 Accessing the register set in Thumb state

In Thumb state, there are fewer instructions than in ARM state to access the Program Status Registers and the high registers (r8-r15):

- In Thumb state, there are no MRS or MSR instructions to move data between the CPSR or SPSRs and the general-purpose registers.
- In Thumb state, only the following instructions can access the high registers:
  - the ADD (4) form of the ADD instruction
  - the CMP (3) form of the CMP instruction
  - the MOV (3) form of the MOV instruction
  - the BLX (2) form of the BLX instruction
  - the BX instruction.

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

### 2.4.2 Program Status Registers

The processor has one *Current Program Status Register* (CPSR), and five *Saved Program Status Registers* (SPSRs) for exception handlers to use. The Program Status Registers:

- hold information about the most recently performed ALU operation
- control the enabling and disabling of interrupts
- set the processor operating mode.

Figure 2-2 shows the bit fields in the Program Status Registers.

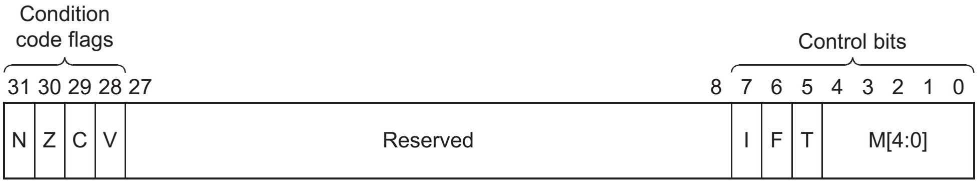


**Figure 2-2 Program Status Registers**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Note**

For compatibility with future ARM processors, do not alter the reserved bits of a Program Status Register. Use read-modify-write operations when changing the CPSR.

Table 2-2 describes the bit fields of the Program Status Registers.

**Table 2-2 Program Status Register encoding**

| Bit | Name | Definition |
|---|---|---|
| [31] | N | Overflow flag:<br>1 = overflow in last operation<br>0 = no overflow. |
| [30] | Z | Zero flag:<br>1 = result of 0 in last operation<br>0 = nonzero result. |
| [29] | C | Carry/borrow flag:<br>1 = carry or borrow in last operation<br>0 = no carry or borrow. |
| [28] | V | Negative or less than flag:<br>1 = result negative or less than in last operation<br>0 = result positive or greater than. |
| [27:8] | - | Reserved |
| [7] | I | IRQ disable bit:<br>1 = IRQ interrupts disabled<br>0 = IRQ interrupts enabled. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Table 2-2 Program Status Register encoding (continued)**

| Bit | Name | Definition |
|---|---|---|
| [6] | F | FIQ disable bit:<br>1 = FIQ interrupts disabled<br>0 = FIQ interrupts enabled. |
| [5] | T | Thumb state flag:<br>1 = processor operating in Thumb state<br>0 = processor operating in ARM state. |
| [4:0] | M | Mode field:<br>b10000 = User mode<br>b10001 = FIQ mode<br>b10010 = IRQ mode<br>b10011 = Supervisor mode<br>b10111 = Abort mode<br>b11011 = Undefined mode<br>b11111 = System mode. |

**Note**

Writing a value to M[4:0] that is not listed in Table 2-2 on page 2-8 causes the processor to enter an unrecoverable state. If this occurs, apply Reset.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 2.5 Data types

The ARM968E-S processor supports the following data types:

- 32-bit words
- 16-bit halfwords
- 8-bit bytes.

You must align the data as follows:

- align words to four-byte boundaries
- align halfwords to two-byte boundaries
- align bytes to byte boundaries.

**Note**

Memory systems are expected to support all data types. In particular, the system must support subword writes without corrupting neighboring bytes in that word.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 2.6 Memory formats

The ARM968E-S processor views memory as a linear collection of bytes numbered in ascending order from 0. For example:

- bytes 0-3 hold the first stored word
- bytes 4-7 hold the second stored word.

The processor can treat words in memory as being stored in:

- little-endian format
- big-endian format.

——— **Note** ———

Little-endian is the default memory format for ARM processors.

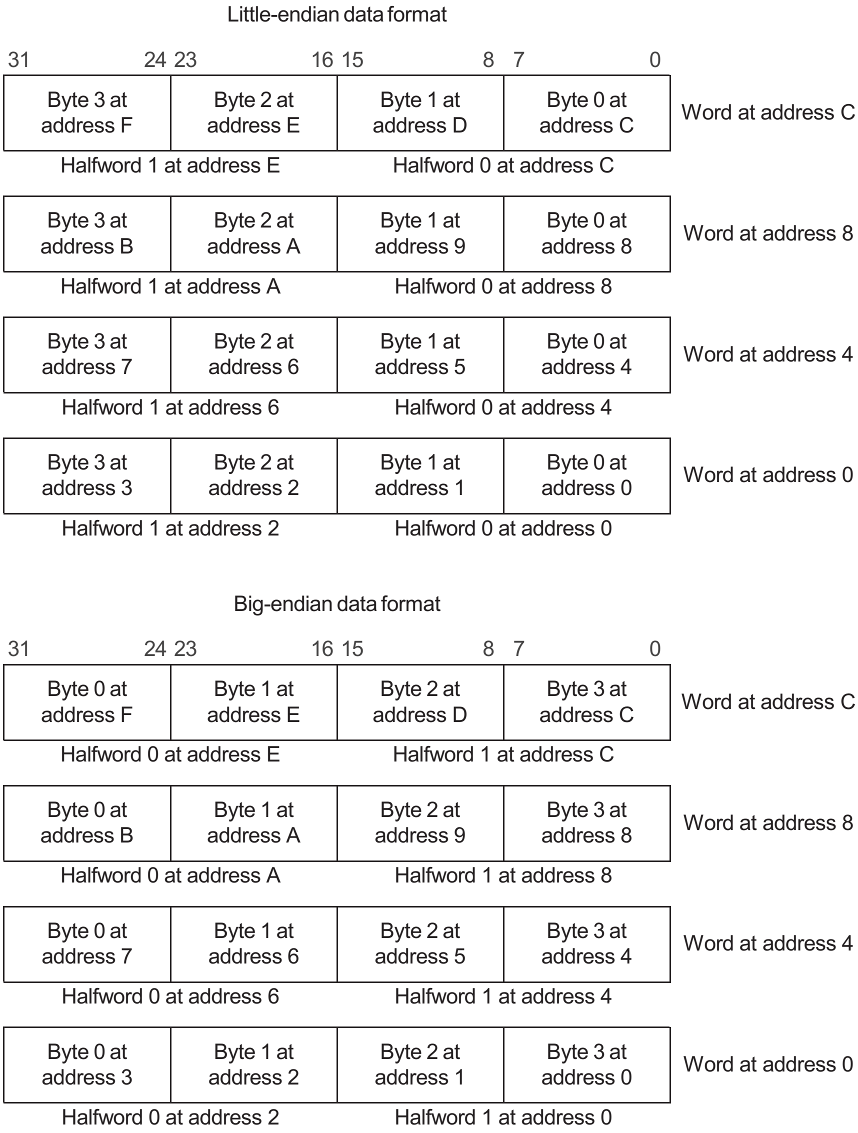
———————————

In little-endian format, the byte with the lowest address in a word is the least-significant byte of the word. The byte with the highest address in a word is the most significant byte. The byte at address 0 of the memory system connects to data lines 7-0.

In big-endian format, the byte with the lowest address in a word is the most significant byte of the word. The byte with the highest address in a word is the least significant byte. The byte at address 0 of the memory system connects to data lines 31-24.

Figure 2-3 on page 2-12 shows the difference between little-endian and big-endian memory formats.

Copyright © 2004, 2006 ARM Limited. All rights reserved.



Little-endian data format

| 31 24 | 23 16 | 15 8 | 7 0 | |
|---|---|---|---|---|
| Byte 3 at address F | Byte 2 at address E | Byte 1 at address D | Byte 0 at address C | Word at address C |
| Halfword 1 at address E | | Halfword 0 at address C | | |
| Byte 3 at address B | Byte 2 at address A | Byte 1 at address 9 | Byte 0 at address 8 | Word at address 8 |
| Halfword 1 at address A | | Halfword 0 at address 8 | | |
| Byte 3 at address 7 | Byte 2 at address 6 | Byte 1 at address 5 | Byte 0 at address 4 | Word at address 4 |
| Halfword 1 at address 6 | | Halfword 0 at address 4 | | |
| Byte 3 at address 3 | Byte 2 at address 2 | Byte 1 at address 1 | Byte 0 at address 0 | Word at address 0 |
| Halfword 1 at address 2 | | Halfword 0 at address 0 | | |

Big-endian data format

| 31 24 | 23 16 | 15 8 | 7 0 | |
|---|---|---|---|---|
| Byte 0 at address F | Byte 1 at address E | Byte 2 at address D | Byte 3 at address C | Word at address C |
| Halfword 0 at address E | | Halfword 1 at address C | | |
| Byte 0 at address B | Byte 1 at address A | Byte 2 at address 9 | Byte 3 at address 8 | Word at address 8 |
| Halfword 0 at address A | | Halfword 1 at address 8 | | |
| Byte 0 at address 7 | Byte 1 at address 6 | Byte 2 at address 5 | Byte 3 at address 4 | Word at address 4 |
| Halfword 0 at address 6 | | Halfword 1 at address 4 | | |
| Byte 0 at address 3 | Byte 1 at address 2 | Byte 2 at address 1 | Byte 3 at address 0 | Word at address 0 |
| Halfword 0 at address 2 | | Halfword 1 at address 0 | | |

**Figure 2-3 Little-endian and big-endian memory formats**

## 2.7 Exceptions

Exceptions occur whenever the sequential flow of a program has to be temporarily changed. For example, the program flow can change to service an interrupt from a peripheral device. Before attempting to handle the exception, the processor preserves the current processor state so that it can return to the original flow after handling the exception.

If two or more exceptions occur simultaneously, the exceptions are dealt with in the fixed order given in *Exception priorities* on page 2-18.

The following sections describe ARM968E-S exception handling:

- *Entering an exception*
- *Exiting an exception* on page 2-17.

### 2.7.1 Entering an exception

When handling an ARM exception, the processor performs the following sequence of operations:

- preserves the address of the next instruction in the appropriate LR:
  - in ARM state, the processor copies the current PC + 4 or PC + 8 value to the LR (see Table 2-3 on page 2-17)
  - in Thumb state, the processor copies the current PC + 2, PC + 4, or PC + 8 value to the LR (see Table 2-3 on page 2-17).

  **Note**

  The exception handler does not have to determine the processor state when entering an exception. For example, in the case of an SWI in either ARM state or Thumb state, the following instruction returns to the next instruction:

  ```
  MOVS PC, r14_svc
  ```

- copies the CPSR into the appropriate SPSR
- forces the CPSR mode bits to a value that depends on the exception type
- forces the PC to fetch the next instruction from the appropriate exception vector.

The processor can also set the interrupt disable bits to prevent unmanageable nesting of exceptions.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Note**

The ARM968E-S processor always enters, handles, and exits exceptions in ARM state. If the processor is in Thumb state when an exception occurs, it automatically switches to ARM state when it loads the exception vector address into the PC. The exception handler might change to Thumb state, but it must return to ARM state to enable the exception handler to terminate correctly.

### Reset

Driving the **HRESETn** signal LOW generates a Reset, and the ARM968E-S processor stops executing the current instruction. When **HRESETn** returns to a HIGH state, the ARM968E-S processor:

- forces the CPSR M[4:0] field to b10011 to enter Supervisor mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR F bit to disable FIQ interrupts
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the Reset vector address.

### Abort

An abort occurs when the memory system cannot complete a data access or an instruction prefetch as described in the following sections:

- *Data Abort*
- *Prefetch Abort* on page 2-15.

#### *Data Abort*

When the memory system signals a Data Abort, the ARM968E-S processor:

- marks the loaded or stored data as invalid
- enters the Data Abort exception before any following instructions alter the processor state
- writes the address of the aborted instruction into r14_abt
- copies the contents of the CPSR into the SPSR_abt
- forces the CPSR M[4:0] field to b10111 to enter Abort mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the Data Abort vector address.

#### *Prefetch Abort*

When the memory system signals a Prefetch Abort, the ARM968E-S processor:

- marks the fetched instruction as invalid
- enters the Prefetch Abort exception when the instruction reaches the Execute stage of the pipeline
- writes the address of the aborted instruction into r14_abt
- copies the contents of the CPSR into the SPSR_abt
- forces the CPSR M[4:0] field to b10111 to enter Abort mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the Data Abort vector address.

### Fast interrupt request

An FIQ is a fast interrupt caused by a LOW level on the **nFIQ** input. The **nFIQ** input passes into the processor through a synchronizer. When FIQ interrupts are enabled, the processor checks for a LOW level on the output of the FIQ synchronizer at the end of each instruction.

When an FIQ interrupt occurs, the ARM968E-S processor:

- writes the address of the next instruction to be executed plus four into r14_fiq
- copies the contents of the CPSR into the SPSR_fiq
- forces the CPSR M[4:0] field to b10001 to enter FIQ mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR F bit to disable FIQ interrupts
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the FIQ vector address.

FIQs and IRQs are disabled when an FIQ occurs. You can use nested interrupts, but you must save any corruptible registers and re-enable FIQ and IRQ interrupts.

### Interrupt request

The IRQ exception is a normal interrupt caused by a LOW level on the **nIRQ** input. An IRQ interrupt has lower priority than an FIQ interrupt. The processor disables IRQ interrupts when it enters an FIQ sequence.

When an FIQ interrupt occurs, the ARM968E-S processor:

- writes the address of the next instruction to be executed plus four into r14_irq
- copies the contents of the CPSR into the SPSR_irq
- forces the CPSR M[4:0] field to b10010 to enter IRQ mode

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

- clears the CPSR T bit to enter ARM state
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the IRQ vector address.

The processor disables IRQ interrupts when it enters an IRQ sequence. You can use nested interrupts, but you must save any corruptible registers and re-enable IRQ interrupts.

### Undefined instruction

When the processor encounters an instruction that neither it nor any coprocessor in the system can handle, it takes the Undefined instruction trap. Software can use this mechanism to extend the ARM instruction set by emulating Undefined coprocessor instructions.

When an Undefined instruction exception occurs, the ARM968E-S processor:

- writes the address of the next instruction to be executed into r14_und
- copies the contents of the CPSR into the SPSR_und
- forces the CPSR M[4:0] field to b11011 to enter Undefined mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the Undefined vector address.

### Software interrupt instruction

You can use the SWI instruction to enter Supervisor mode to request a particular supervisor function. The SWI handler reads the opcode to extract the SWI function number.

When an SWI instruction is executed, the ARM968E-S processor:

- writes the address of the next instruction to be executed into r14_svc
- copies the contents of the CPSR into the SPSR_svc
- forces the CPSR M[4:0] field to b10011 to enter Supervisor mode
- clears the CPSR T bit to enter ARM state
- sets the CPSR I bit to disable IRQ interrupts
- forces the PC to the SWI vector address.

### 2.7.2 Exiting an exception

When exception processing is completed, the exception handler must perform the following steps:

1. Move the LR value, minus an offset, to the PC. The offset depends on the type of exception, as Table 2-3 shows.

**Table 2-3 Exception return points**

| | Saved LR value | | | |
|---|---|---|---|---|
| **Exception** | **ARM state** | **Thumb state** | **Return instruction** | **Return point** |
| Reset | - | - | - | After Reset, r14_svc value is Unpredictable |
| Data Abort | PC + 8 | PC + 8 | `SUBS PC, R14_abt, #8` | Aborted instruction |
| | | | `SUBS PC, R14_abt, #4` | Instruction after aborted instruction |
| FIQ | PC + 4 | PC + 4 | `SUBS PC, R14_fiq, #4` | Interrupted instruction |
| IRQ | PC + 4 | PC + 4 | `SUBS PC, R14_irq, #4` | Interrupted instruction |
| Prefetch Abort | PC + 4 | PC + 4 | `SUBS PC, R14_abt, #4` | Aborted instruction |
| Undefined instruction | PC + 4 | PC + 2 | `MOVS PC, R14_und` | Instruction after Undefined instruction |
| SWI instruction | PC + 4 | PC + 2 | `MOVS PC, R14_svc` | Instruction after SWI instruction |

2. Copy the SPSR back to the CPSR.

3. Clear the interrupt disable bits that were set when the processor entered the exception.

**Note**

Restoring the CPSR from the SPSR automatically restores the I, F, and T bits to the values they held immediately before the exception.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 2.7.3 Exception vectors

Table 2-4 lists the exception vector addresses.

**Table 2-4 Exception vectors**

| Exception | Vector address |
|---|---|
| Reset | `0x00000000` |
| Undefined instruction | `0x00000004` |
| SWI | `0x00000008` |
| Prefetch Abort | `0x0000000C` |
| Data Abort | `0x00000010` |
| Reserved | `0x00000014` |
| IRQ | `0x00000018` |
| FIQ | `0x0000001C` |

### 2.7.4 Exception priorities

When multiple exceptions are present, a fixed priority system determines the order in which they are handled. Table 2-5 lists the exception priorities.

**Table 2-5 Exception priorities**

| Priority | Exception |
|---|---|
| Highest | Reset |
| | Data Abort |
| | FIQ |
| | IRQ |
| | Prefetch Abort |
| Lowest | Undefined instruction and SWI |

Some exceptions cannot occur together:

- The Undefined instruction and SWI exceptions are mutually exclusive. Each corresponds to a particular, non-overlapping, decoding of the current instruction.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

- When FIQs are enabled, and a Data Abort occurs at the same time as an FIQ, the processor enters the Data Abort handler, and proceeds immediately to the FIQ vector.

  A normal return from the FIQ causes the Data Abort handler to resume execution.

  Data Aborts must have higher priority than FIQs to ensure that the transfer error does not escape detection. You must add the time for this exception entry to the worst-case FIQ latency calculations in a system that uses aborts to support virtual memory.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 3
# Memory Map

This chapter describes the ARM968E-S processor fixed memory map implementation. It contains the following sections:

- *About the ARM968E-S memory map* on page 3-2
- *Tightly-coupled memory address space* on page 3-3
- *Bufferable write address space* on page 3-4.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 3.1 About the ARM968E-S memory map

The fixed-size *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) enable high-speed operation without incurring the performance and power penalties of accessing the system bus. Write buffers decouple the processor from wait states incurred when accessing the AHB bus and the TCMs.

The fixed memory map provides simple control over the TCM and AHB write buffers. Figure 3-1 shows the ARM968E-S memory map.

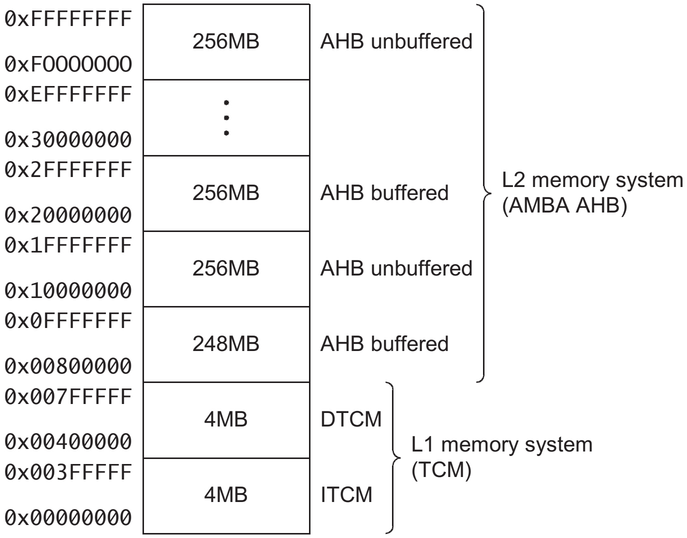


**Figure 3-1 ARM968E-S memory map**

## 3.2 Tightly-coupled memory address space

The TCM space is at the bottom of the memory map. The memory map allocates the bottom 4MB for the ITCM and the next 4MB for the DTCM.

In practice, each TCM is likely to be much smaller than 4MB. The address decode for ARM968E-S processor accesses and DMA controller accesses is implemented so that each memory is aliased throughout its 4MB range. Figure 3-2 shows an example of a 16KB ITCM aliased through the 4MB ITCM address space.

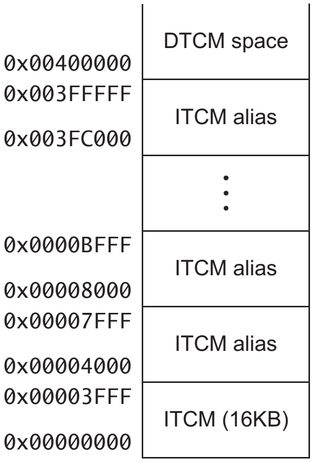


**Figure 3-2 ITCM aliasing example**

All ARM968E-S processor accesses to addresses above the 8MB of combined TCM address space result in AMBA AHB transfers controlled by the *Bus Interface Unit* (BIU).

**Note**

All DMA accesses to addresses above 8MB are aliased back into the 0MB-8MB address space.

All instruction fetches to the DTCM address space go to the AHB. A data interface access from the processor can access both the DTCM and the ITCM. The ability to access the ITCM with the data interface is required for fetching inline literals within code, for programming of the ITCM, and for debugging.

When a TCM is disabled, all accesses to its address space go to the AHB. When enabled, the TCM must be programmed before use. The value of the input pin **INITRAM** during Reset enables or disables the TCMs. Several boot options are available using **INITRAM** and the exception vectors location pin **VINITHI**. *Enabling TCM* on page 6-4 describes these options.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 3.3 Bufferable write address space

The use of the AHB write buffer is controlled by both the CP15 c1 Control Register and the fixed address map.

When the processor comes out of Reset, the AHB write buffer is disabled. All data writes to the AHB are performed as unbuffered. The processor can stall until the BIU completes the write on the AHB interface.

When the AHB write buffer is enabled by setting bit 3 of the CP15 c1 Control Register, the data address (**DA[31:0]**) from the processor determines if the AHB write buffer is used. If **DA[28]** is set, the write is unbuffered. If **DA[28]** is clear, the write is buffered and uses the AHB write buffer. Buffered writes enable the processor to continue program execution while the write is performed on the AHB. If the AHB write buffer is full, the processor stalls until space in the buffer becomes available. See *AHB write buffer* on page 5-9 for descriptions of the BIU and AHB write buffer behavior.

**Note**

Writes to TCM address space do not write through to the AHB if the TCM being accessed is enabled.

Writes to the address space of a disabled TCM are buffered AHB writes when the AHB write buffer is enabled or unbuffered AHB writes when the AHB write buffer is not enabled.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 4
# System Control Coprocessor

This chapter describes how to use the cp15 registers in the System Control Coprocessor to configure the ARM968E-S processor and to control and monitor its operation. It contains the following sections:

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 4.1 About the System Control Processor

You can use the System Control Coprocessor to:

- read the 32-bit ID code of the ARM968E-S processor
- read the sizes of the ITCM and the DTCM
- enable:
  - address alignment fault checking
  - big-endian or little-endian memory mapping
  - the AHB write buffer
  - the *Instruction Tightly-Coupled Memory* (ITCM) and *Data Tightly-Coupled Memory* (DTCM)
  - low vector location or high vector location
  - entry into Thumb state on PC loads.
- perform wait-for-interrupt and drain-write-buffer operations
- read the trace process ID
- stall ITCM and DTCM accesses while the AHB write buffer is full, enable or disable the instruction prefetch buffer, and mask IRQ or FIQ interrupts while the ETM FIFO is full.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 4.2 Accessing CP15 registers

You can access the CP15 registers only with MCR and MRC instructions and only in a privileged mode. Figure 4-1 shows the format of CP15 MCR and MRC instructions.

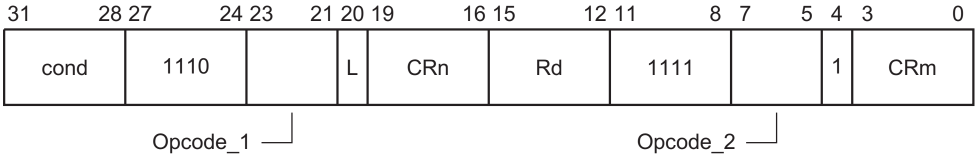


**Figure 4-1 CP15 MCR and MRC instruction format**

The assembly code for these instructions is:

```
MCR{cond} p15, <opcode_1>, <Rd>, <CRn>, <CRm>, <opcode_2>
MRC{cond} p15, <opcode_1>, <Rd>, <CRn>, <CRm>, <opcode_2>
```

Accessing CP15 registers in User mode or with the coprocessor instructions CDP, LDC, and STC in any processor mode generates the Undefined Instruction exception.

See the *ARM Architecture Reference Manual* for a description of the MCR and MRC instructions.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 4.3 CP15 register summary

Table 4-1 lists the CP15 registers.

**Table 4-1 CP15 register summary**

| Name | Access | Reset value | Description |
|---|---|---|---|
| CP15 c0 Device ID Register | Read-only | `0x41059680` | See *CP15 c0 Device ID Register* on page 4-5 |
| CP15 c0 TCM Size Register | Read-only | Implementation-defined[a] | See *CP15 c0 TCM Size Register* on page 4-6 |
| CP15 c1 Control Register | Read/write | Implementation-defined[b] | See *CP15 c1 Control Register* on page 4-7 |
| CP15 c7 core control operations | Write-only | - | See *CP15 c7 core control operations* on page 4-9 |
| CP15 c13 Trace Process ID Register | Read/write | `0x00000000` | See *CP15 c13 Trace Process ID Register* on page 4-11 |
| CP15 c15 Configuration Control Register | Read/write | `0x00000004` | See *CP15 c15 Configuration Control Register* on page 4-11 |

[a] Value at Reset determined by ITCM size and DTCM size. See Table 4-3 on page 4-6.

[b] Value at Reset determined by INITRAM and VINITHI pins. See Table 4-5 on page 4-8.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 4.4 CP15 register descriptions

This section describes the CP15 registers.

- *CP15 c0 Device ID Register*
- *CP15 c0 TCM Size Register* on page 4-6
- *CP15 c1 Control Register* on page 4-7
- *CP15 c7 core control operations* on page 4-9
- *CP15 c13 Trace Process ID Register* on page 4-11
- *CP15 c15 Configuration Control Register* on page 4-11.

### 4.4.1 CP15 c0 Device ID Register

Use the read-only Device ID Register to read the 32-bit ID code of the ARM968E-S processor.

Read the Device ID Register with the following instruction:

```
MRC p15, 0, <Rd>, c0, c0, {0, 1, 3-7}; read Device ID Register
```

When reading the Device ID Register, the opcode_2 field can be any value other than 2. Writing to the Device ID Register is Unpredictable.

Figure 4-2 shows the bit fields of the Device ID Register.

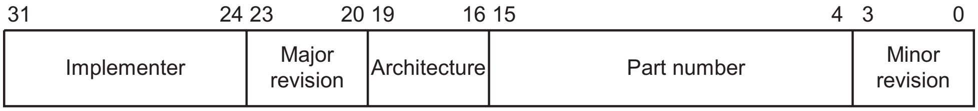


**Figure 4-2 Device ID Register**

Table 4-2 describes the bit fields of the Device ID Register.

**Table 4-2 Encoding of the Device ID Register**

| Bit | Name | Definition | Value |
|---|---|---|---|
| [31:24] | Implementer | Implementer's trademark<br>ARM Limited uses the ASCII code for the letter A, `0x41` | `0x41` |
| [23:20] | Major revision | Major specification revision (0) | `0x0` |
| [19:16] | Architecture | ARM architecture version (v5TE) | `0x5` |
| [15:4] | Part number | Processor number (968) | `0x968` |
| [3:0] | Minor revision | Minor specification revision (1) | `0x1` |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 4.4.2 CP15 c0 TCM Size Register

Use the read-only TCM Size Register to read the sizes of the ITCM and the DTCM.

Read the TCM Size Register with the following instruction:

```
MRC p15, 0, <Rd>, c0, c0, 2; read TCM Size Register
```

Writing to the TCM Size Register is Unpredictable.

Figure 4-3 shows the bit fields of the TCM Size Register.

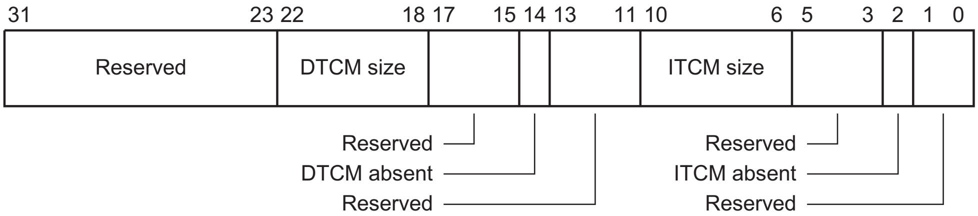


**Figure 4-3 TCM Size Register**

Table 4-3 describes the bit fields of the TCM Size Register.

**Table 4-3 Encoding of the TCM Size Register**

| Bit | Name | Definition |
|---|---|---|
| [31:23] | - | Reserved. |
| [22:18} | DTCM size | DTCM size bits:<br>b00000 = 0KB<br>b00001 = 1KB<br>b00010 = 2KB<br>b00011 = 4KB<br>b00100 = 8KB<br>b00101 = 16 KB<br>b00110 = 32KB<br>b00111 = 64KB<br>b01000 = 128KB<br>b01001 = 256KB<br>b01010 = 512KB<br>b01011 = 1MB<br>b01100 = 2MB<br>b01101 = 4MB.<br>At Reset, the **DTCMSIZE[4:0]** pins determine the value of the DTCM size field. |
| [17:15] | - | Reserved. |
| [14] | DTCM absent | Set when **DTCMSIZE[4:0]** = b00000.<br>At Reset, the **DTCMSIZE[4:0]** pins determine the value of the DTCM absent bit. |
| [13:11] | - | Reserved. |

**Table 4-3 Encoding of the TCM Size Register (continued)**

| Bit | Name | Definition |
|---|---|---|
| [10:6] | ITCM size | ITCM size bits:<br>b00000 = 0KB b00111 = 64KB<br>b00001 = 1KB b01000 = 128KB<br>b00010 = 2KB b01001 = 256KB<br>b00011 = 4KB b01010 = 512KB<br>b00100 = 8KB b01011 = 1MB<br>b00101 = 16 KB b01100 = 2MB<br>b00110 = 32KB b01101 = 4MB.<br>At Reset, the **ITCMSIZE[4:0]** pins determine the value of the ITCM size field. |
| [5:3] | - | Reserved. |
| [2] | ITCM absent | Set when **ITCMSIZE[4:0]** = b00000.<br>At Reset, the **ITCMSIZE[4:0]** pins determine the value of the ITCM absent bit. |
| [1:0] | - | Reserved. |

### 4.4.3 CP15 c1 Control Register

Use the read/write Control Register to:

- prevent PC loads from changing the T bit
- select high-address or low-address vector locations
- enable the ITCM and DTCM
- select big-endian or little-endian operation
- enable the AHB write buffer
- enable fault checking of address alignment.

Access the Control Register with the instructions in Table 4-4.

**Table 4-4 Control Register instructions**

| Instruction | Operation |
|---|---|
| `MRC p15, 0, <Rd>, c1, c0, 0` | Read Control Register |
| `MCR p15, 0, <Rd>, c1, c0, 0` | Write Control Register |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Figure 4-4 shows the Control Register bit fields.

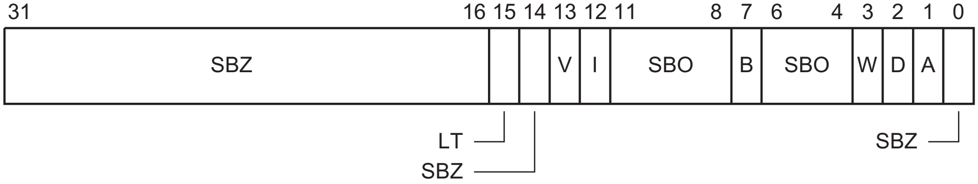


**Figure 4-4 Control Register**

Table 4-5 describes the bit fields of the Control Register.

**Table 4-5 Encoding of the Control Register**

| Bit | Name | Definition |
|---|---|---|
| [31:16] | - | Should Be Zero. |
| [15] | LT | Load Thumb disable bit:<br>1 = loading PC does not set T bit<br>0 = loading PC sets T bit.<br>Reset clears the LT bit. |
| [14] | - | Should Be Zero. |
| [13] | V | Exception vector location bit:<br>1 = vector address range is `0xFFFF0000` to `0xFFFF001C`<br>0 = vector address range is `0x00000000` to `0x0000001C`.<br>At Reset, the **VINITHI** pin determines the value of the V bit. You can write to V after Reset. |
| [12] | I | ITCM enable bit:<br>1 = instruction accesses use ITCM interface<br>0 = instruction accesses use AHB interface.<br>At Reset, the **INITRAM** pin determines the value of the I bit. You can write to I after Reset. |
| [11:8] | - | Should Be One. |
| [7] | B | Big-endian bit:<br>1 = big-endian memory mapping<br>0 = little-endian memory mapping.<br>Reset clears the B bit. |
| [6:4] | - | Should Be One. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Table 4-5 Encoding of the Control Register (continued)**

| Bit | Name | Definition |
|---|---|---|
| [3] | W | AHB write buffer enable bit:<br>1 = write buffer enabled<br>0 = write buffer disabled.<br>Clearing the W bit causes AHB write buffer entries to complete as buffered writes. Reset clears W. |
| [2] | D | DTCM enable bit:<br>1 = data accesses use DTCM interface<br>0 = data accesses use AHB interface.<br>At Reset, the **INITRAM** pin determines the value of the D bit. |
| [1] | A | Address alignment fault checking enable bit:<br>1 = fault checking of address alignment enabled<br>0 = fault checking of address alignment disabled.<br>Reset clears the A bit. |
| [0] | - | Should Be Zero. |

### 4.4.4 CP15 c7 core control operations

Use CP15 c7 core control operations to:

- put the processor in the low-power wait-for-interrupt state
- drain the AHB and TCM write buffers.

Perform the core control operations with the instructions in Table 4-6.

**Table 4-6 Core control instructions**

| Instruction | Operation |
|---|---|
| `MCR p15, 0, <Rd>, c7, c0, 4` | Enter wait-for-interrupt state |
| `MCR p15, 0, <Rd>, c7, c10, 4` | Drain write buffers |

**Note**

For compatibility with existing software, the ARM968E-S processor also supports the following instruction for the wait-for-interrupt function:

```
MCR p15, 0, <Rd>, c15, c8, 2
```

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Wait-for-interrupt operation

The wait-for-interrupt operation drains the AHB write buffer and puts the ARM968E-S processor in a low-power standby state. The processor clock stops from the time that the wait-for-interrupt instruction is executed until **nFIQ**, **nIRQ**, or **EDBGRQ** is asserted.

**Note**

Asserting **nIRQ** or **nFIQ** wakes the processor from wait-for-interrupt state even if the I or F interrupt disable bit is set in the CPSR. If interrupts are enabled, the ARM968E-S processor is guaranteed to take the interrupt before executing the next instruction after the wait-for-interrupt instruction.

When debug is enabled, setting the debug request bit in the EmbeddedICE-RT Control Register also terminates the wait-for-interrupt state. The processor enters debug state before executing any more instructions.

Entering wait-for-interrupt mode asserts the **STANDBYWFI** signal. You can use **STANDBYWFI** to shut down clocks to other system blocks that do not have to be clocked when the ARM968E-S processor is idle.

The DMA interface has a separate clock enable, **HCLKEND**, that enables the DMA interface to continue operating while the ARM968E-S is in wait-for-interrupt mode. This feature enables you to use the external DMA controller to transfer data to and from the TCM before waking the processor.

### Drain-write-buffer operation

The drain-write-buffer operation acts as an explicit instruction memory barrier. It stalls instruction execution until the AHB and TCM write buffers are emptied. This operation is useful in real-time applications when a write to a peripheral must be completed before program execution continues. For example, it might be necessary to drain the write buffers when a peripheral in a bufferable region is the source of an interrupt. After interrupt servicing, the request must be removed before interrupts can be enabled again. This can be ensured by separating the store to the peripheral and the interrupt enable function with a drain-write-buffers operation.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

### 4.4.5 CP15 c13 Trace Process ID Register

The read/write Trace Process ID Register enables the real-time trace tools to identify the currently executing process in multitasking environments. Use the instructions in Table 4-7 to access the Trace Process ID Register.

**Table 4-7 Trace Process ID Register instructions**

| Instruction | Operation |
|---|---|
| `MRC p15, 0, <Rd>, c13, {c0-c15}` | Read Trace Process ID Register |
| `MCR p15, 0, <Rd>, c13, {c0-c15}` | Write Trace Process ID Register |

Figure 4-5 shows the bit field of the Trace Process ID Register.

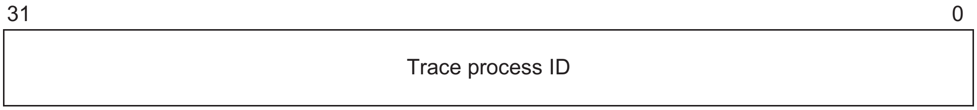


**Figure 4-5 Trace Process ID Register**

The **ETMPROCID[31:0]** pins reflect the contents of the Trace Process ID Register. Reset clears the Trace Process ID Register.

**Note**

Writing to the Trace Process ID Register sets the **ETMPROCIDWR** signal for one clock cycle.

### 4.4.6 CP15 c15 Configuration Control Register

Use the read/write Configuration Control Register to:

- stall ITCM or DTCM accesses when the ITCM or DTCM write buffer contains data
- disable the *Instruction Prefetch Buffer* (IPB)
- mask FIQ or IRQ interrupts when the ETM FIFO is full.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Access the Configuration Control Register with the instructions in Table 4-8.

**Table 4-8 Configuration Control Register instructions**

| Instruction | Operation |
|---|---|
| `MRC p15, 1, <Rd>, c15, c1, 0` | Read Configuration Control Register |
| `MCR p15, 1, <Rd>, c15, c1, 0` | Write Configuration Control Register |

Figure 4-6 shows the bit fields of the Configuration Control Register.

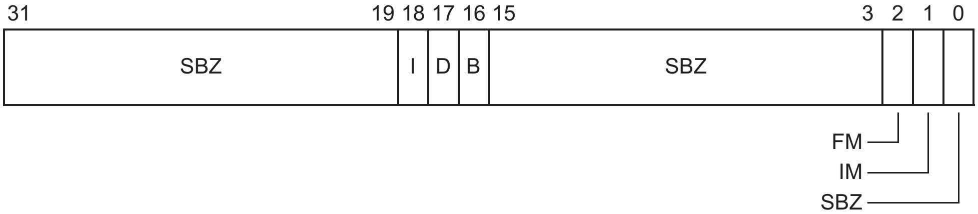


**Figure 4-6 Configuration Control Register**

Table 4-9 describes the bit fields of the Configuration Control Register.

**Table 4-9 Encoding of the Configuration Control Register**

| Bit | Name | Definition |
|---|---|---|
| [31:19] | - | Should Be Zero. |
| [18] | I | ITCM order bit:<br>1 = ITCM accesses stalled if ITCM write buffer contains data<br>0 = ITCM accesses not stalled by data in ITCM write buffer.<br>Asserting the I bit ensures that TCM accesses are performed in the order generated by the processor and that writes are committed to memory before subsequent reads are done. Asserting I when data is still in the TCM write buffer stalls any subsequent TCM access until the buffer is empty. Reset clears the I bit. |
| [17] | D | DTCM order bit:<br>1 = DTCM accesses stalled if TCM write buffer contains data<br>0 = DTCM accesses not stalled by data in TCM write buffer.<br>Asserting the D bit ensures that TCM accesses are performed in the order generated by the processor and that writes are committed to memory before subsequent reads are done. Asserting D when data is still in the TCM write buffer stalls any subsequent TCM access until the buffer is empty. Reset clears the D bit. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Table 4-9 Encoding of the Configuration Control Register (continued)

| Bit | Name | Definition |
|---|---|---|
| [16] | B | AHB instruction prefetch buffer disable bit:<br>1 = instruction prefetch buffer disabled<br>0 = instruction prefetch buffer enabled.<br>When the B bit is set, all instruction accesses are performed as nonsequential transfers. See *Instruction prefetch buffer* on page 5-5. Reset clears B. |
| [15:3] | - | Should Be Zero. |
| [2] | FM | FIQ interrupt mask when ETM FIFO is full:<br>1 = **nFIQ** cannot enable processor clocks when **FIFOFULL** is HIGH<br>0 = **nFIQ** enables processor clocks until interrupt is serviced, but clocks are disabled on exit from FIQ mode. Reset sets the FM bit. |
| [1] | IM | IRQ interrupt mask when ETM FIFO is full:<br>1 = **nIRQ** cannot enable processor clocks when **FIFOFULL** is HIGH<br>0 = **nIRQ** enables processor clocks until interrupt is serviced, but clocks are disabled on exit from IRQ mode. Reset clears the IM bit. |
| [0] | - | Should Be Zero. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 4.5 CP15 instruction summary

Table 4-10 is a quick reference to the CP15 instructions.

**Table 4-10 CP15 instruction summary**

| Instruction | Operation | Reference |
|---|---|---|
| `MRC p15, 0, <Rd>, c0, c0, {0, 1, 3-7}` | Read Device ID Register | page 4-5 |
| `MRC p15, 0, <Rd>, c0, c0, 2` | Read TCM Size Register | page 4-6 |
| `MRC p15, 0, <Rd>, c1, c0, 0` | Read Control Register | page 4-7 |
| `MCR p15, 0, <Rd>, c1, c0, 0` | Write Control Register | |
| `MCR p15, 0, <Rd>, c7, c0, 4` | Enter wait-for-interrupt state | page 4-9 |
| `MCR p15, 0, <Rd>, c15, c8, 2` | Enter wait-for-interrupt state | |
| `MCR p15, 0, <Rd>, c7, c10, 4` | Drain write buffers | |
| `MRC p15, 0, <Rd>, c13, {c0-c15}` | Read Trace Process ID Register | page 4-11 |
| `MCR p15, 0, <Rd>, c13, {c0-c15}` | Write Trace Process ID Register | |
| `MRC p15, 1, <Rd>, c15, c1, 0` | Read Configuration Control Register | page 4-11 |
| `MCR p15, 1, <Rd>, c15, c1, 0` | Write Configuration Control Register | |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 5
# Bus Interface Unit

This chapter describes the ARM968E-S *Bus Interface Unit* (BIU) and AHB write buffer. It contains the following sections:

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 5.1 About the BIU

The ARM968E-S processor uses the *Advanced Microprocessor Bus Architecture* (AMBA) *Advanced High-performance Bus-Lite* (AHB-Lite) interface. The AHB-Lite version of the AMBA interface addresses the requirements of synthesizable high-performance designs, including:

- single rising-clock-edge operation
- unidirectional buses
- mapped burst transfers.

See the *AMBA Specification* (Rev 2.0) for a full description of this bus architecture.

The BIU implements a fully-compliant AHB-Lite bus master interface with an *Instruction Prefetch Buffer* (IPB) and an AHB write buffer to increase system performance. The BIU is the link between the processor's *Tightly-Coupled Memories* (TCMs) and the external AHB memory. The AHB memory or the DMA must be used to initialize the TCMs and to access code and data that are not assigned to the TCM address space.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 5.2 Bus transfer characteristics

The BIU handles all data transfers and instruction transfers between the core clock domain and the AMBA bus clock domain. Any request from the IPB or the AHB write buffer that has to go outside the ARM968E-S processor is handled by the bus interface in a way that is transparent to the processor.

The types of AMBA bus transfers are:

- noncachable instruction fetches and data loads
- nonbuffered data stores
- buffered data stores
- noncachable nonbufferable data swap operations.

Each of the AMBA AHB bus transfers generates a signature. Table 5-1 lists the types of BIU transfers and their characteristics.

**Table 5-1 BIU transfer characteristics**

| Transfer | HADDR[a] | HTRANS[b] | HPROT[c] | HSIZE | HBURST | HLOCK | HWRITE | HRDATA/ HWDATA |
|---|---|---|---|---|---|---|---|---|
| Instruction fetch | [31:0] | NS-S-S- . . . -S | [0 0 p 0] | 32 | Incr[d] | 0 | 0 | [31:0] |
| Noncachable load | [31:0] | NS | [0 b p t] | 8, 16, 32 | Single/Incr[d] | 0 | 0 | [31:0] |
| Nonbufferable store | [31:0] | NS-S-S- . . . -S | [0 0 p 1] | 8, 16, 32 | Incr[d] | 0 | 1 | [31:0] |
| Buffered store | [31:2] bb | NS-S-S- . . . -S | [0 1 p 1] | 8, 16, 32 | Incr[d] | 0 | 1 | [31:0] |
| Swap (load) | [31:2] 00 | NS | [0 0 p 1] | 32 | Single | 1 | 0 | [31:0] |
| Swap (store) | [31:2] 00 | NS | [0 0 p 1] | 32 | Single | 1 | 1 | [31:0] |

a. See *Transfer size*.

b. See *Sequential and nonsequential transfers* on page 5-4.

c. See *BIU protection control* on page 5-4.

d. Incr burst type covers INCR, INCR4, INCR8, and INCR16.

### 5.2.1 Transfer size

**HSIZE[1:0]** defines transfer size and determines values of low-order address bits **HADDR[1:0]**, that appear in the **HADDR** column of Table 5-1 as b or bb. An eight-bit transfer does not affect **HADDR[1:0]**. A 16-bit transfer forces **HADDR[0]** to 0. A 32-bit transfer forces **HADDR[1:0]** to b00.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 5.2.2 Sequential and nonsequential transfers

The HTRANS column in Table 5-1 on page 5-3 shows that transfers are *Sequential* (S) or *NonSequential* (NS). Any burst of four elements is always an NS-S-S-S transfer. Any burst of eight elements is always an NS-S-S-S-S-S-S-S transfer. For BIU bursts listed in Table 5-1 on page 5-3 that support an incrementing burst type, the burst can be 1, 4, 8, or 16 elements, shown as NS-S-S- . . . -S.

### 5.2.3 BIU protection control

The four HPROT column in Table 5-1 on page 5-3 shows the four protection attributes:

- cachability = 0 for all ARM968E-S transfers.
- bufferability:
  - data load or instruction fetch = 0
  - data stores = 0 (nonbufferable) or 1 (bufferable).
- accessibility:
  - User mode = 0
  - privileged mode = 1.
- transfer type:
  - instruction fetch = 0
  - data access = 1.

### 5.2.4 BIU locked transfers

The BIU can perform locked bus transfers only for ARM swap instructions. It begins the swap operation by asserting **HLOCK** and performing a locked nonsequential read.

The BIU keeps **HLOCK** asserted until the ARM968E-S processor performs the nonsequential write. Until the nonsequential write begins, the BIU issues idle AHB cycles. During the idle BIU AHB cycles, **HSIZE**, **HADDR**, **HBURST**, **HPROT**, and **HWRITE** hold their values.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 5.3 Instruction prefetch buffer

The *Instruction Prefetch Buffer* (IPB) is four 32-bit entries deep. All nonsequential instruction fetches to AHB space cause the IPB to be flushed and an initial burst of four words to be performed on the AHB. After this initial burst, the IPB performs AHB accesses to keep the buffer full. If the processor takes an instruction out of the buffer on each clock cycle, the fetches on the AHB interface are performed as incrementing bursts of unspecified length (**HBURST[2:0]** = b001).

Only valid instruction requests initiate prefetching. The IPB marks each entry with the error response returned from the AHB. The IPB also marks each entry with the external breakpoint request returned from the external memory system.

Instruction prefetching does not cross 1KB boundaries.

### 5.3.1 Optimized Thumb instruction prefetch

In Thumb state, the IPB depth is reduced to two words (four Thumb instructions). The processor performs a two-word incrementing burst for nonsequential fetches. When space becomes available, the IPB performs transfers to fill any vacant entries up to the buffer depth limit of two word entries available in Thumb state.

### 5.3.2 IPB disable bit

Setting bit 16 of the CP15 c15 Configuration Control Register disables the IPB. See *CP15 c15 Configuration Control Register* on page 4-11. Reset clears bit 16 and enables prefetching.

### 5.3.3 AHB error response in IPB

If an error response is returned from the AHB, it is stored in the IPB along with the instruction. If the instruction reaches the Execute stage of the pipeline, a Prefetch Abort exception occurs.

### 5.3.4 IPB timing examples

This section gives two examples of IPB operation:

- *Nonsequential instruction fetch* on page 5-6
- *Nonsequential instruction fetch after a data access* on page 5-7.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Nonsequential instruction fetch

Figure 5-1 shows AHB prefetching in operation. In this case, the processor is executing code sequentially from the AHB. When each instruction returns from the AHB, it is returned to the processor. Because the processor is continuously requesting instructions from the AHB, none of the returned data is placed into the IPB. The AHB runs ahead of the processor to minimize the number of stall cycles.

The processor then generates a nonsequential instruction fetch. This can be a result of a branch or an operation changing the value of the PC. The BIU terminates the current burst and starts a new prefetch operation with a burst of length four to fill the prefetch buffer. The first instruction from the burst is returned to the processor. The BIU keeps the IPB full by performing a burst of undefined length. This is because the processor is running sequentially and requesting instructions each cycle. Because this is a new burst, AHB indicates NSEQ.

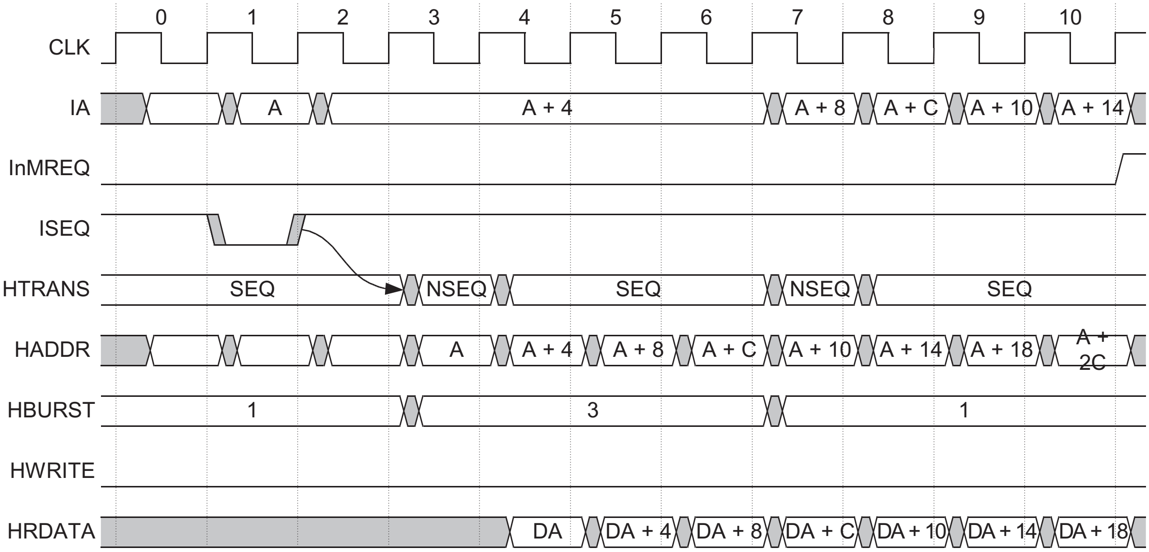


**Figure 5-1 Nonsequential instruction fetch**

**Note**

All timing examples in this chapter are based on one-to-one clocking in which the processor and AHB share the same clock. See *AHB clocking* on page 5-17 for details of AHB clocking modes.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Nonsequential instruction fetch after a data access

Figure 5-2 on page 5-8 shows an AHB data access between instruction fetches. Because data accesses take precedence over instruction fetches, the instruction fetch starts after the data access. After the first instruction address is issued on the AHB, sequential instruction prefetching starts. The core does not advance until both of the simultaneous memory requests are satisfied.

As in the previous example in Figure 5-1 on page 5-6, the IPB is not used immediately because each instruction returned from the BIU is immediately used by the processor. The second data memory request causes the processor to stall until the data request is completed. This causes the two outstanding instruction prefetches to be stored in the IPB. Prefetching stops as a result of a data request.

The instruction request issued with the data access can be acknowledged as soon as the AHB transfer is complete. After the data access, prefetching can continue because the address is sequential to the previous instruction address.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

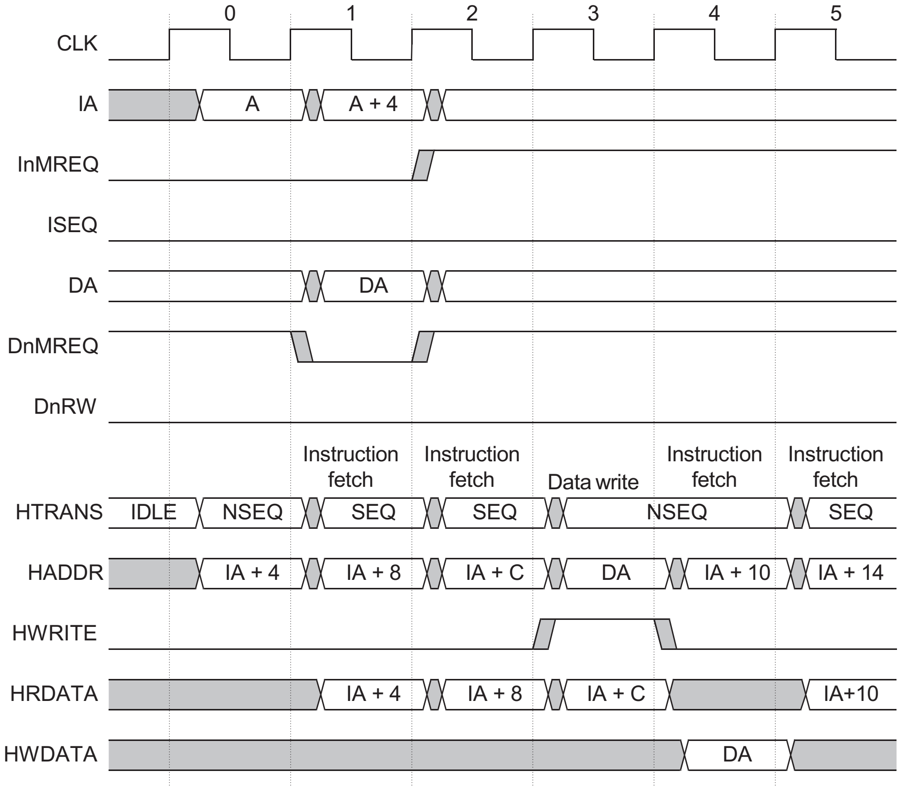


**Figure 5-2 Nonsequential instruction fetch after a data access**

## 5.4 AHB write buffer

The AHB write buffer can hold two addresses and four data words. The write buffer decouples the processor from the wait cycles caused by accessing the AHB. If a write is sent to the write buffer, the processor is able to continue program execution without having to wait for the write to complete on the AHB. If there is room in the write buffer, more writes can be committed to the write buffer without stalling. If the processor tries to write to a buffered location when the write buffer is full, the processor stalls until there is space in the write buffer.

If the processor performs a read from AHB address space or an unbuffered write to AHB address space, the read stalls until all write buffer entries are written. Draining the write buffer ensures data coherency.

### 5.4.1 Committing write data to the AHB write buffer

The AHB write buffer is used when the following conditions are met:

- the write buffer is enabled
- the write address is in a bufferable region
- the write address is in AHB address space
- the write address selects a tightly-coupled memory that is disabled.

For details on enabling the AHB write buffer and about the fixed address map, see

- *CP15 c1 Control Register* on page 4-7
- *About the ARM968E-S memory map* on page 3-2.

When the processor performs a write that conforms to these conditions, the address for the write is put into the address entry of the AHB write buffer FIFO. The next available entry data is used for the write data. If the write is a store multiple (STM), subsequent data entries are used for each word of the STM. It is therefore possible for the FIFO to contain four words of an STM.

Alternatively, if several shorter bufferable STM or single write (STR) instructions are performed, one address entry is used for each write instruction. The worst case is that only one data word fills the FIFO caused by one STR write. In this case, the FIFO holds one address entry and one data entry.

### 5.4.2 Draining write data from the AHB write buffer

The AHB write buffer can drain naturally when AHB writes occur each time data is committed to the FIFO. The processor stalls only if the write buffer overflows. However, there are times when a complete drain of the write buffer is enforced.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

### Natural AHB write buffer drain

When a write is being committed to the AHB write buffer, a signal to the BIU initiates an AHB write. The BIU then pops the address for the write from the write buffer followed by the data and starts an AHB transfer. This process might take several cycles because the write access is to an AHB slave in a bufferable region that has a multicycle response. If the AHB is running at a lower rate than the processor, there can be extra delay in the buffered write process. This can cause the processor to fill the write buffer by committing data faster than the write buffer can drain. Then the processor stalls until an entry becomes available.

Placing an address in the AHB write buffer stores a marker with the address to indicate that the size of the write is byte, halfword, or word. An STM operation stores a sequentiality marker with the data. The sequentiality marker indicates that the BIU must use the address incrementor to produce the AHB addresses for the second and following writes of the STM. These markers enable the write buffer to store an address with only one FIFO entry, leaving more room for data.

### Enforced AHB write buffer drain

There are two situations in which the processor stalls and the AHB write buffer is forced to drain completely before program execution can continue:

- the processor requests an instruction fetch, data load, or unbuffered AHB write
- the processor performs a drain-write-buffer operation.

### AHB read access requested

To ensure data coherency, the processor must be prevented from reading a location when new data for that location is still in the AHB write buffer. If the read occurs before the write buffer is drained, the processor reads the old data, causing a data coherency failure.

For this reason, whenever an AHB load or instruction fetch is requested, the processor must be stalled until the write buffer is drained. There is no dedicated logic to initiate a write buffer drain. However, there is dedicated logic that stalls the processor until the last buffered write is completed on the AHB.

### Drain-write-buffer operation

You can use an MCR instruction to CP15 c7 to stall the processor until the AHB write buffer is empty. This operation is described in *CP15 c7 core control operations* on page 4-9. This instruction is useful when a write must be completed before program execution can continue.

### 5.4.3 Enabling the AHB write buffer

Setting bit 3 of the CP15 c1 Control Register enables the AHB write buffer. When bit 3 is set, all writes to bufferable address locations use the write buffer. If a slave peripheral in a bufferable region returns an AHB Data Abort, the abort is ignored when the write buffer is enabled.

——— **Note** ———

For debugging purposes, you can disable the AHB write buffer to enable AHB Data Aborts to be returned from bufferable regions.

———

### 5.4.4 Disabling the AHB write buffer

When data is committed to the AHB write buffer, it is always written to the AHB. Disabling the write buffer by clearing bit 3 of the CP15 c1 Control Register causes any existing data in the write buffer to be written. Performing the wait-for-interrupt operation also causes any data in the write buffer to be written.

To prevent buffered writes after disabling the write buffer or after the wait-for-interrupt operation, first perform the drain-write-buffer operation.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 5.5 AHB bus master interface

The ARM968E-S processor has the AHB-Lite bus master interface. See the *AMBA Specification (Rev 2.0)* for a detailed description of the AHB protocol.

### 5.5.1 Overview of AHB

The AHB architecture is based on separate cycles for address and data. The address and control values for an access are broadcast from the rising edge of **HCLK** in the cycle before the data is expected to be read or written. During this data cycle, the address and control values for the next cycle are driven out. This leads to a fully-pipelined address architecture.

When an access is in its data cycle, a slave can wait the access by driving the **HREADY** response LOW. This has the effect of stretching the current data cycle and the pipelined address and control for the next access. This creates a system in which all AHB masters and slaves sample **HREADY** on the rising edge of the **HCLK** to determine if an access is complete and a new address can be sampled or driven out.

## 5.6 AHB transfer descriptions

The ARM968E-S BIU performs a subset of the possible AHB bus transfers. This section describes the transfers that can be performed and some back-to-back transfer cases:

- *Back-to-back data transfers*
- *Data burst crossing a 1KB boundary* on page 5-15
- *SWP instruction* on page 5-15.

### 5.6.1 Back-to-back data transfers

Figure 5-3 shows bus activity when a sequence of STR instructions is executed with no AHB instruction fetches. The processor is executing instructions from the TCM space.

In cycle 1 the processor starts a nonsequential data write. A series of nonsequential and idle transfers is indicated for each access. The processor is re-enabled in cycle 9.

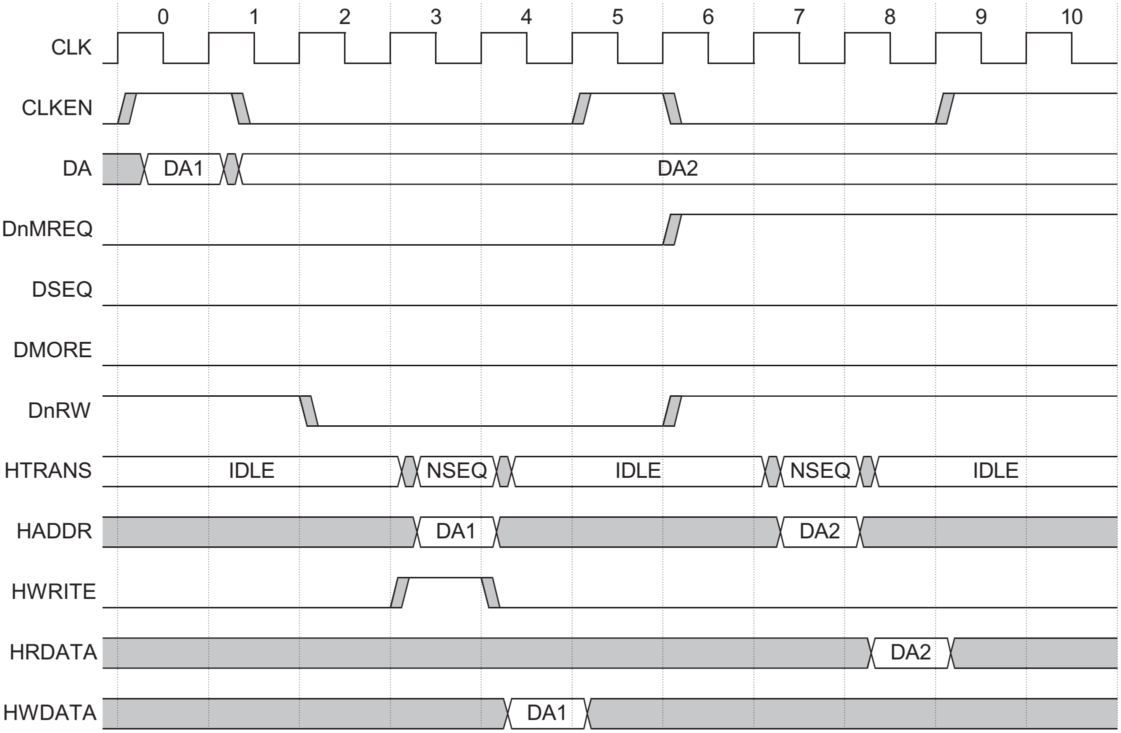


**Figure 5-3 Back-to-back writes followed by a read**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Note**

Executing a sequence of back-to-back LDR instructions produces the same series of nonsequential and idle transfers.

### STM followed by instruction fetch

Figure 5-4 shows an example of an STM transferring four words, immediately followed by an instruction fetch. The instruction read begins with a nonsequential/sequential sequence after the final sequential data access. In this example, subsequent instruction fetches are sequential. Instruction prefetching is enabled so that instruction fetches appear on the AHB before the processor requests them.

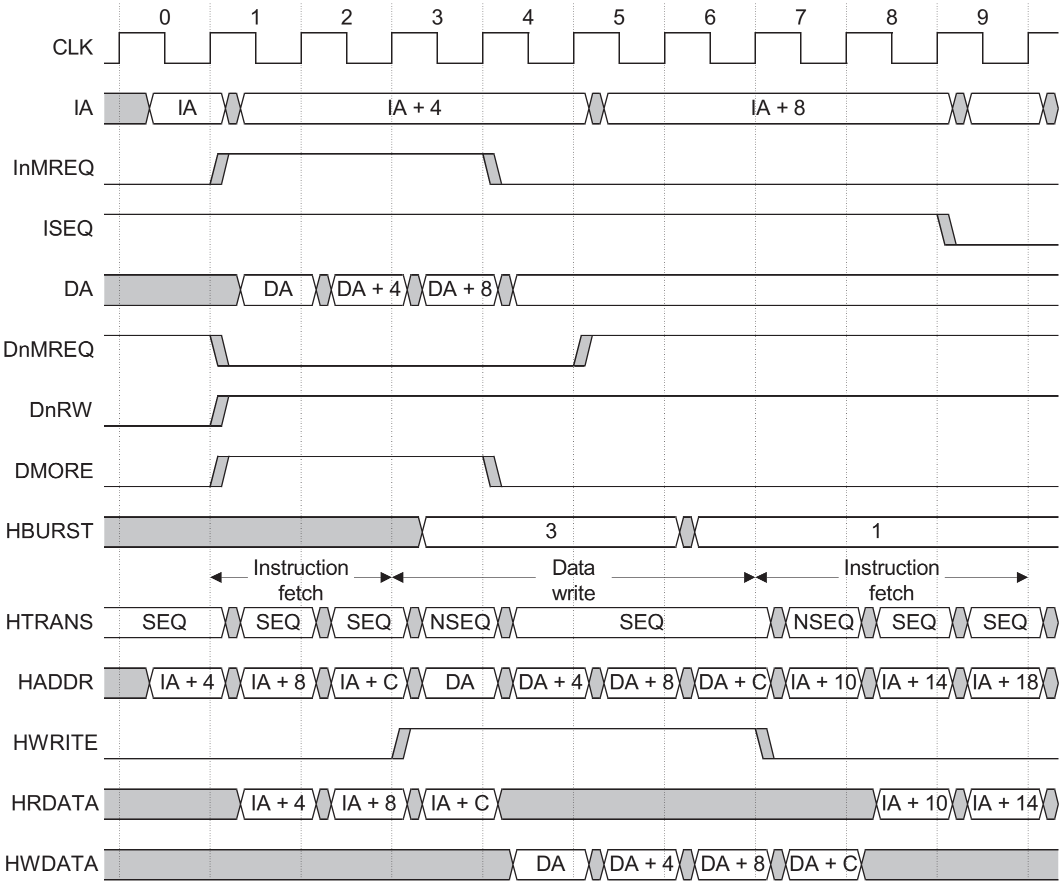


**Figure 5-4 Single STM followed by sequential instruction fetch**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Data burst crossing a 1KB boundary

The *AMBA Specification (Rev 2.0)* states that sequential accesses must not cross 1KB boundaries. The processor splits sequential accesses that cross a 1KB boundary into two sets of separate accesses.

Figure 5-5 shows bus activity with two back-to-back STM instructions crossing a 1KB boundary. DA + 8 is the first address in a new 1KB region. The two sets of transfers begin with a nonsequential access and are separated by idle cycles. In this example, instructions are being fetched from the ITCM.

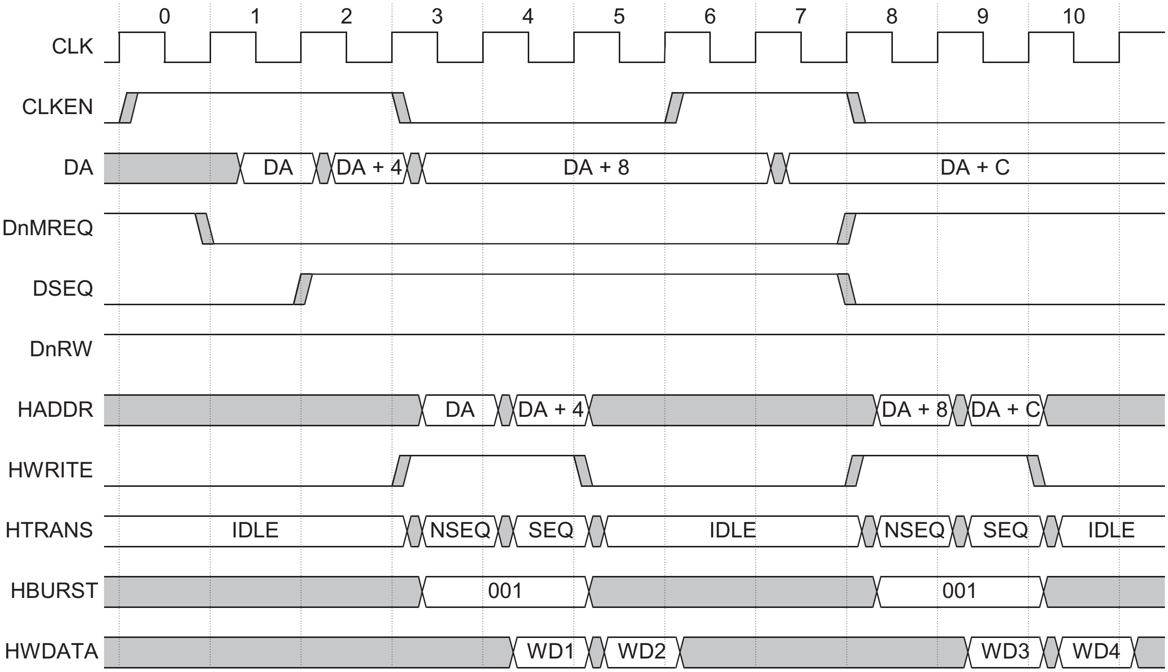


**Figure 5-5 Data burst crossing a 1KB boundary**

### SWP instruction

The SWP instruction performs an atomic read-modify-write operation. It is commonly used with semaphores to guarantee that another process cannot modify a semaphore when it is being read by the current process.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

If the processor performs an SWP operation to an AHB address location, the access is always unbuffered to ensure that the processor stalls until the write occurs on the AHB. The BIU asserts the **HMASTLOCK** output to prevent the AHB arbiter from providing ownership to a different master, ensuring that the read-modify-write is atomic. In the example in Figure 5-6, instructions are being fetched from the ITCM.

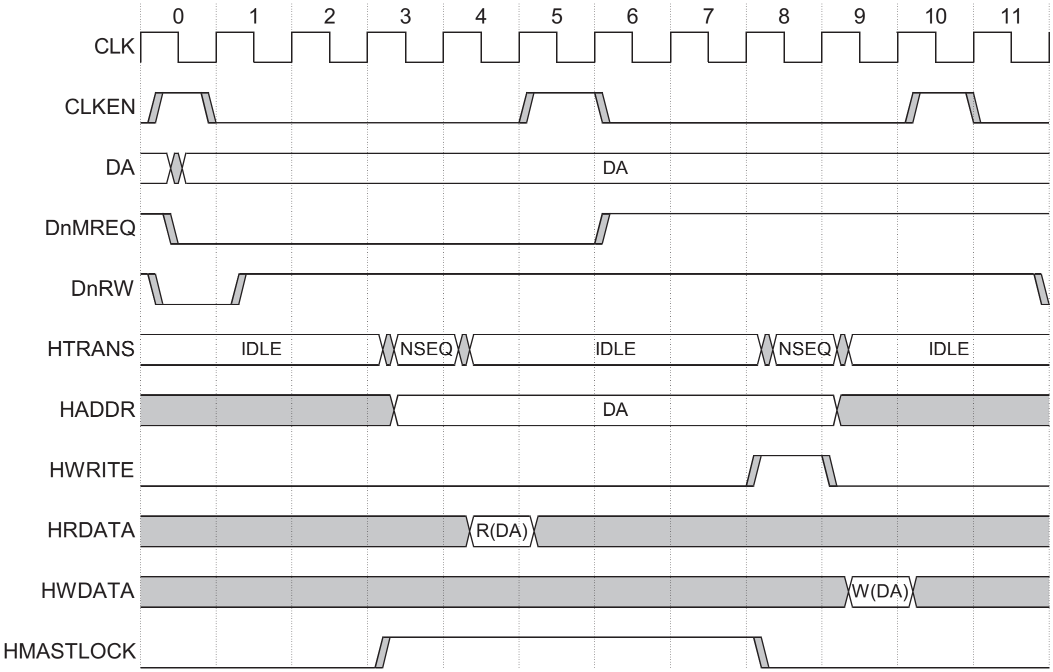


**Figure 5-6 SWP instruction**

### 5.6.2 Data burst support

To enable more efficient use of burst-capable memories on the AHB bus, the BIU detects processor data burst sizes that align with AHB incrementing burst sizes. The processor supports incrementing burst sizes of 4, 8, and 16 words. These bursts are performed on the bus as defined-length bursts. Bursts that cross 1K boundaries are split into two separate transactions on the AHB, one on each side of the boundary. All bursts that do not map onto AHB incrementing burst sizes are marked as unspecified length.

## 5.7 AHB clocking

The ARM968E-S processor uses a single rising-edge clock signal, **CLK**, to time all internal activity. In a system with an embedded processor, it can be best to run the AHB at a lower clock rate. To support a lower AHB clock rate, the processor must have a clock enable, **HCLKEN**, to time AHB transfers.

The **HCLKEN** input is driven HIGH around a rising edge of **CLK** to indicate that this rising edge is also a rising edge of **HCLK**. **HCLK** must therefore be synchronous to **CLK**.

When the processor is running from TCM or performing writes using the AHB write buffer, the **HCLKEN** and **HREADY** inputs are decoupled from the **SYSCLKEN** stall signal. The processor is stalled only by TCM stall cycles or if the write buffer overflows. This means that the processor is executing instructions at the faster **CLK** rate and is decoupled from the **HCLK** domain AHB system.

If however, an AHB read or unbuffered write is required, the processor stalls until the AHB transfer is complete. Because the AHB system is being clocked by the slower **HCLK**, the processor must examine **HCLKEN** to detect when to drive out the AHB address and control signals to start an AHB transfer. **HCLKEN** then has to detect the following rising edges of **HCLK** so that the BIU can detect when the access completes. Figure 5-7 shows an example of an AHB read with a 3:1 ratio of **CLK** to **HCLK**.

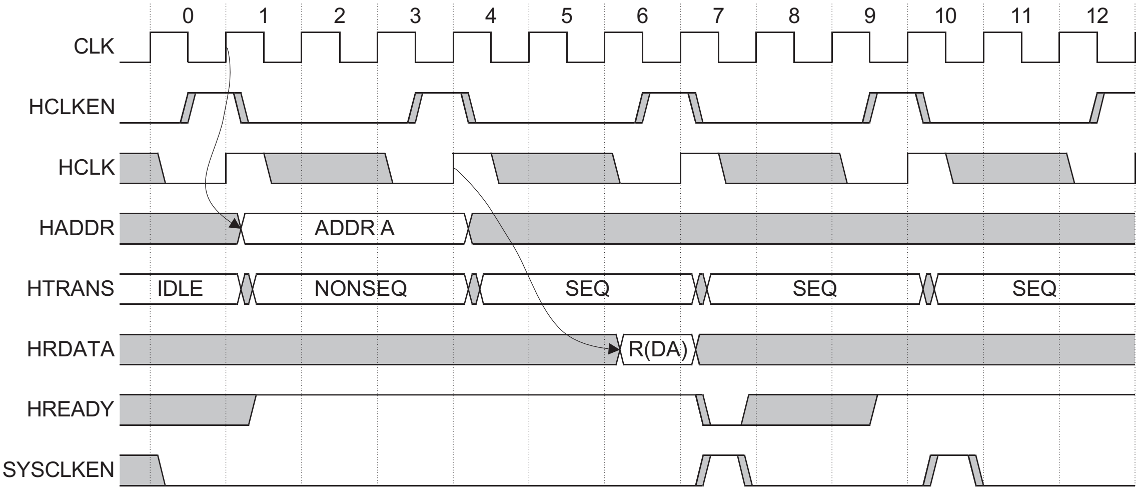


**Figure 5-7 AHB 3:1 clocking example**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

If the slave being accessed at the **HCLK** rate has a multicycle response, the **HREADY** input to the processor is driven LOW until the data is ready to be returned. The BIU must therefore perform a logical AND of the **HREADY** response and **HCLKEN** to detect that the AHB transfer has completed. When the AND is true, the processor is then enabled by reasserting **SYSCLKEN**.

**Note**

Before the processor can start an AHB access, it must wait until it receives the next **HCLKEN** pulse. Then it must wait until the access is complete. The stall before the start of the access is a synchronization penalty, and the worst case can be expressed in **CLK** cycles as the **CLK**-to-**HCLK** ratio minus one.

## 5.8 CLK-to-HCLK skew

The ARM968E-S processor drives out the AHB address on the rising edge of **CLK** when the **HCLKEN** input is true. The AHB outputs have output hold and delay values relative to **CLK**. However, these outputs are used in the AHB system where **HCLK** is used to time the transfers. Similarly, inputs to the processor are timed relative to **HCLK** but are sampled within the processor with **CLK**. Minimizing the skew between **HCLK** and **CLK** prevents hold time issues from **CLK** to **HCLK** on outputs and from **HCLK** to **CLK** on inputs.

### 5.8.1 Clock tree insertion at top level

The processor clock tree enables an evenly distributed clock to be driven to all the registers in the design. As Figure 5-8 on page 5-20 shows, the registers that drive AHB outputs and sample AHB inputs are timed off **CLK’** at the bottom of the inserted clock tree and are subject to the clock tree insertion delay. When the processor is embedded in an AHB system, the clock generation logic to produce **HCLK** must be constrained so that it matches the insertion delay of the clock tree within the processor. This can easily be done by performing a top-level clock tree insertion for the processor and the embedded system at the same time.

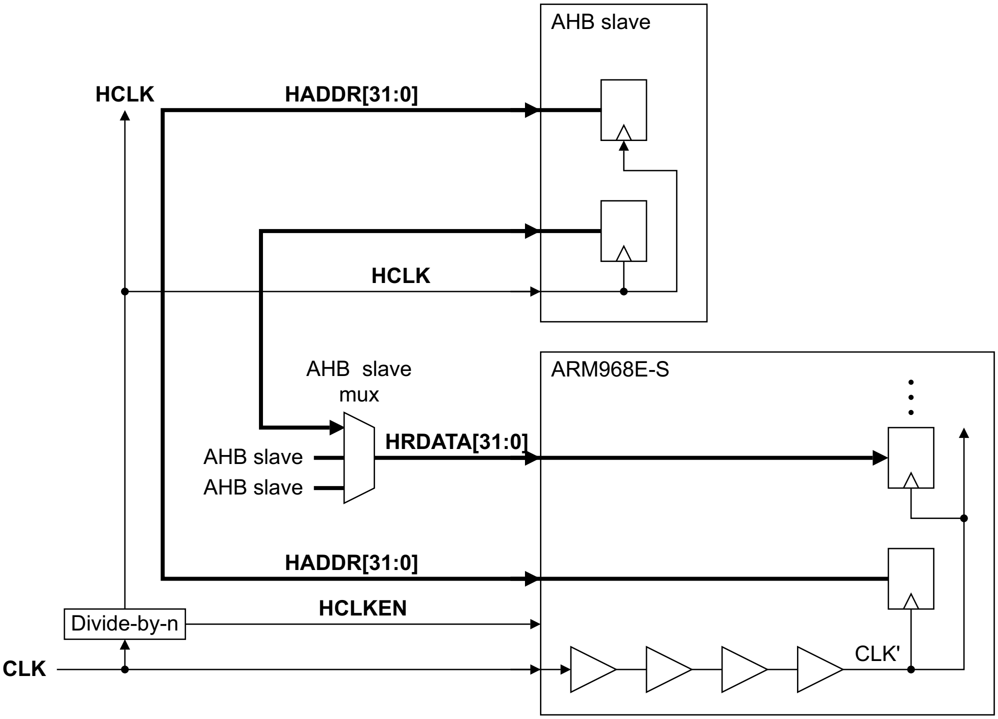


**Figure 5-8 CLK to HCLK sampling**

In this example, the slave peripheral has an input setup and hold time and an output hold and valid time relative to **HCLK**. The ARM968E-S processor has an input setup and hold time and an output hold and valid relative to **CLK'**, the clock at the bottom of the clock tree. For optimal performance, clock tree insertion must be used to balance **HCLK** to match **CLK'**.

### 5.8.2 Hierarchical clock tree insertion

If clock tree insertion is performed before embedding the processor, buffers are added on input data to match the clock tree so that the setup and hold is relative to the top level **CLK**. This is guaranteed to be safe at the expense of extra buffers in the data input path. However, inserting the buffers can limit operating frequency.

The **HCLK** domain AHB peripherals must still meet the ARM968E-S input setup and hold requirements. Because the processor inputs and outputs are now relative to **CLK**, the outputs appear comparatively later by the value of the insertion delay. This ultimately leads to lower AHB operating frequency.

Chapter 6
# Tightly-Coupled Memory Interface

This chapter describes the interface for the *Data and Instruction Tightly-Coupled Memory* (DTCM and ITCM). It contains the following sections:

## 6.1 About the TCM interface

The ARM968E-S processor supports both instruction and data TCMs. TCM accesses are deterministic and do not access the AHB. Therefore, you can use the DTCM and ITCM to store real-time, performance-critical code.

The features of the TCM interface include:

- independent ITCM and DTCM sizes of 0KB or 1KB-4MB in power-of-two increments
- alternately accessed DTCM ports, D0TCM and D1TCM, for simultaneous, interleaved DMA and processor access to DTCM at 32-bit (word) granularity
- software visibility and programmability of TCM size and enable
- boot control for ITCM
- data access to the ITCM for literal pool generation in code
- simple SRAM-style interface supporting both reads and writes
- variable TCM wait state control for ITCM and DTCM
- separate AHB-Lite slave interface for DMA engine.

The TCM is located in the TCM address space. See Chapter 3 *Memory Map*.

Figure 6-1 on page 6-3 shows the structure of the TCM interface.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

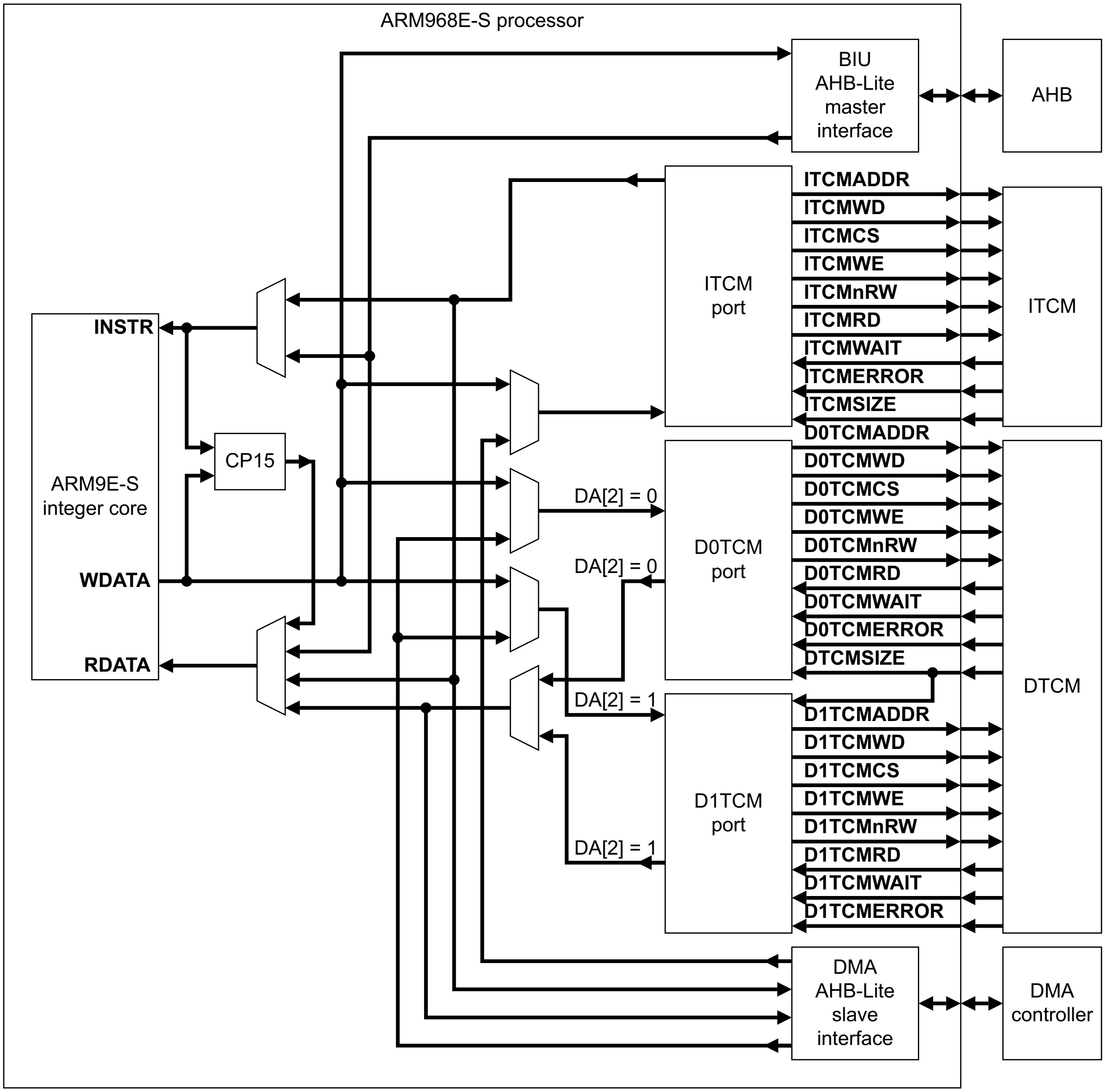


**Figure 6-1 TCM interface**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 6.2 Enabling TCM

This section describes how to use the two mechanisms for controlling the enable of the TCM:

- *Using INITRAM input pin*
- *Using CP15 c1 Control Register* on page 6-5.

### 6.2.1 Using INITRAM input pin

The **INITRAM** pin is provided to enable the ARM968E-S macrocell to boot with both external instruction and data memory blocks either enabled or disabled. Two resets are described in the following sections:

- *Reset with INITRAM LOW*
- *Reset with INITRAM HIGH*.

#### Reset with INITRAM LOW

If **INITRAM** is held LOW during reset, the ARM968E-S macrocell comes out of reset with both external instruction and data memory disabled. All accesses to external instruction and data memory space go to the AHB. The TCMs can then be individually or jointly enabled by writing to the CP15 Control Register.

#### Reset with INITRAM HIGH

If **INITRAM** is held high during reset, both external instruction and data memory are enabled when the ARM968E-S macrocell comes out of reset. This is normally used for a warm reset where the TCM has already been programmed before the application of **HRESETn** to the ARM968E-S macrocell. In this case, the TCM contents are preserved and the ARM968E-S macrocell can run directly from the TCM following reset. Either one or both TCMs can be further disabled or enabled by writing to the CP15 Control Register.

——— **Note** ———

If **INITRAM** is held HIGH during a cold reset (the TCM has not previously been initialized), the **VINITHI** pin must be set HIGH to ensure that the ARM968E-S macrocell boots from `0xFFFF0000`, that is in AHB address space and is substantially outside the TCM address space. This is necessary because if **VINITHI** is LOW, the ARM968E-S macrocell attempts to boot from `0x00000000`, and this selects the uninitialized ITCM.

### 6.2.2 Using CP15 c1 Control Register

When out of Reset, the state of CP15 c1 Control Register determines the behavior of the TCM, as described in the following sections:

- *Enabling the ITCM*
- *Disabling the ITCM*
- *Enabling the DTCM* on page 6-6
- *Disabling the DTCM* on page 6-6.

#### Enabling the ITCM

You can enable the ITCM interface by setting bit [12] of the CP15 c1 Control Register. You must access this register in a read-modify-write fashion to preserve the contents of the bits not being modified. See *CP15 c1 Control Register* on page 4-7 for details of how to read and write the CP15 c1 Control Register. After you enable the ITCM interface, all future ARM9E-S core instruction fetches and data accesses to the ITCM address space cause the ITCM interface to be accessed as shown in Figure 3-1 on page 3-2.

Enabling the ITCM interface greatly increases the performance of the ARM968E-S processor because the majority of accesses to it can be performed with no stall cycles, whereas accessing the AHB might cause several stall cycles for each access. Care must be taken to ensure that the ITCM interface is appropriately initialized before it is enabled and used to supply instructions to the ARM9E-S core. If the core executes instructions from uninitialized ITCM interface, the behavior is Unpredictable.

#### Disabling the ITCM

You can disable the ITCM interface by clearing bit [12] of the CP15 c1 Control Register. After you disable the ITCM interface, all further ARM9E-S core instruction fetches access the AHB. If the core performs a data access to the ITCM address space as shown in Figure 3-1 on page 3-2, an AHB access is performed.

The contents of the memory are preserved when it is disabled. If it is re-enabled, accesses to previously initialized memory locations return the preserved data.

**Note**

The TCM write buffers must be drained before disabling the ITCM interface.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

### Enabling the DTCM

You can enable the DTCM interface by setting bit [2] of the CP15 c1 Control Register. See *CP15 c1 Control Register* on page 4-7 for details of how to read and write this register. After you enable the DTCM interface, all future read and write accesses to the DTCM address space, as shown in Figure 3-1 on page 3-2, cause the DTCM interface to be accessed.

### Disabling the DTCM

You can disable the DTCM by clearing bit [2] of the CP15 c1 Control Register. After you disable the DTCM, all further reads and writes to the DTCM address space, as shown in Figure 3-1 on page 3-2, access the AHB.

**Note**

The TCM write buffers must be drained before disabling the DTCM interface.

## 6.3 TCM write buffers

Each TCM write buffer is two entries deep. Each entry is an address and data pair. In normal operation, the data for a write access to the TCM address space is held in the TCM write buffer until it is forced out by another write to the TCM address space or by natural drain when there are no read requests to the TCM address space.

Write accesses from the processor always go into the TCM write buffer. If there is space in the TCM write buffer, writes are always single-cycle operations regardless of external TCM wait states. If there is no space in the TCM write buffer, any write access stalls the processor until a TCM write buffer entry becomes free.

### 6.3.1 Forcing strict read/write ordering

In normal operation, the TCM write buffer drains naturally into the TCM whenever there are no read accesses to the TCM address space. One effect of this drain mechanism is that read and write accesses to the TCM can be in an order different from that issued by the processor. If the TCM write buffer contains the data required by a read access, data is returned from the buffer. Otherwise, a read can bypass a write that is pending in the TCM write buffer when the read is to a different address.

Read and write accesses to DTCM and ITCM can be maintained in the order that the processor generated them by using the TCM order bits in the CP15 c15 Configuration Control Register. See *CP15 c15 Configuration Control Register* on page 4-11. When the TCM order bit is set, the TCM write buffer is still used but any subsequent read accesses to the TCM are stalled until the buffer is emptied.

To ensure correct operation, perform a drain-write-buffer operation immediately prior to setting the TCM order bit. To drain the TCM and AHB write buffers, use a CP15 c7 core control operation. See *Drain-write-buffer operation* on page 4-10.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 6.4 TCM size

The TCM supports a programmable memory size with a fixed offset defined in the ARM968E-S memory map. Table 6-1 shows how the **ITCMSIZE[4:0]** and **DxTCMSIZE[4:0]** inputs control TCM RAM sizes.

**Table 6-1 Supported TCM RAM sizes**

| TCMSIZE[4:0] inputs | TCM RAM size |
|---|---|
| b00000 | Reserved |
| b00001 | 1KB |
| b00010 | 2KB |
| b00011 | 4KB |
| b00100 | 8KB |
| b00101 | 16KB |
| b00110 | 32KB |
| b00111 | 64KB |
| b01000 | 128KB |
| b01001 | 256KB |
| b01010 | 512KB |
| b01011 | 1MB |
| b01100 | 2MB |
| b01101 | 4MB |
| b01110 | Reserved |
| b01111 | Reserved |
| b1xxxx | Reserved |

The **ITCMSIZE[4:0]** and **DxTCMSIZE[4:0]** inputs are used to ensure that write accesses to aliased addresses return the correct data when read. For correct operation, the **ITCMSIZE[4:0]** and **DxTCMSIZE[4:0]** values must match the instantiated memory size.

The **ITCMSIZE[4:0]** and **DxTCMSIZE[4:0]** inputs are treated as static inputs to the processor, and they must be defined at implementation time. Changing these inputs while the processor is operating results in undefined behavior.

## 6.5 TCM error detection signals

Large SRAM arrays are susceptible to errors caused by alpha particle radiation. These errors can result in incorrect data being returned. You can use parity checking or some form of error detection outside the processor to detect these errors.

To enable the processor to support external error detection on the TCMs, there is one error signal for each of the TCMs:

- **D0TCMERROR** and **D1TCMERROR**
- **ITCMERROR**.

The error signals inform the processor of error conditions during TCM read accesses and are ignored during write accesses. These signals are valid in the same clock cycle as the data returned from the TCM, and the processor ignores them at all other times.

Error detection is performed external to the processor. If error support is not required, **DxTCMERROR** and **ITCMERROR** must be tied LOW. Because the processor is capable of performing byte accesses, parity information must be generated for each byte. The parity bit must be generated at the same time as the data is written to memory. Data is always read from the TCMs in 32-bit words, and a parity error in any byte must be returned to the processor as an ORing of the byte parity and reflected on the **TCMERROR** pins.

For data reads from either the ITCM or DTCM, any error returned causes a Data Abort exception. The exception handler determines what corrective action, if any, to take.

For instruction fetches from the ITCM, any error returned causes a Prefetch Abort exception if the processor tries to execute the returned instruction.

Because of the write buffers in the TCM controller, any read of the TCM by the ARM968E-S processor can result in a partial or full hit in a TCM write buffer. When read results in a full hit in a TCM write buffer, the processor does not generate a request, and **TCMERROR** is ignored. In the case of a partial hit, **TCMERROR** is returned to the processor.

Memory must be initialized to prevent spurious errors.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 6.6 Interface timing

This section gives examples of typical TCM interface transfers:

- *TCM reads with zero wait states*
- *TCM reads with one wait state*
- *TCM reads with four wait states* on page 6-11
- *TCM writes with zero wait states* on page 6-12
- *TCM write with one wait state* on page 6-13
- *TCM write with two wait states* on page 6-13
- *TCM accesses with varying TCM wait states* on page 6-14
- *Speculative TCM read access* on page 6-15.

### 6.6.1 TCM reads with zero wait states

Figure 6-2 is an example of single-cycle TCM read accesses. **ITCMWAIT** is never asserted, and there are no read delays. Read data must be driven in the cycle after the address and TCM control signals are driven.

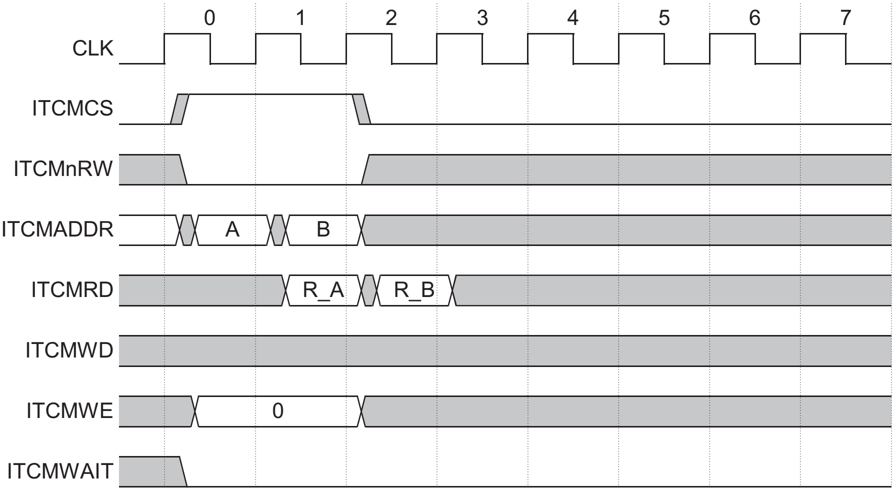


**Figure 6-2 TCM reads with zero wait states**

### 6.6.2 TCM reads with one wait state

Figure 6-3 on page 6-11 is an example of two-cycle TCM read accesses. **ITCMWAIT** delays the R_B and R_C reads for one cycle. Read data must always be driven in the cycle after **ITCMWAIT** is deasserted.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

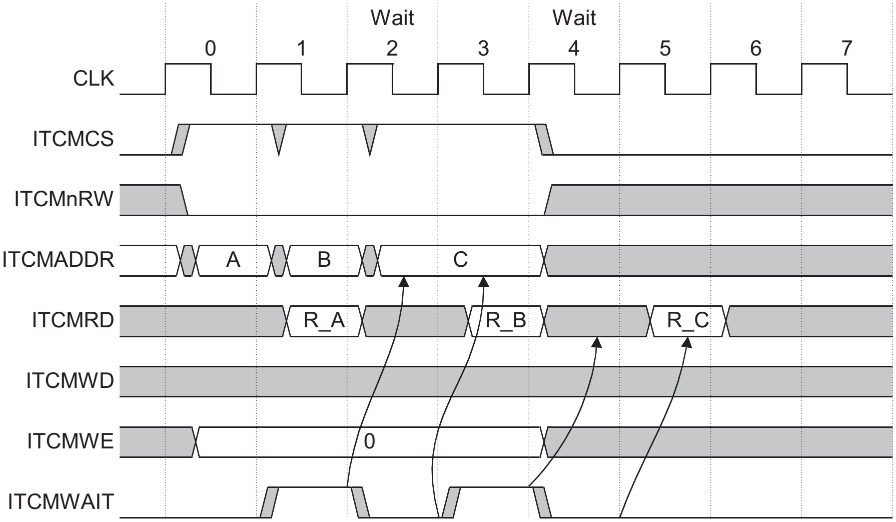


**Figure 6-3 TCM reads with one wait state**

### 6.6.3 TCM reads with four wait states

Figure 6-4 on page 6-12 is an example of a five-cycle TCM read access. **ITCMWAIT** delays the R_B read for four cycles. Read data must always be driven in the cycle after **ITCMWAIT** is deasserted.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

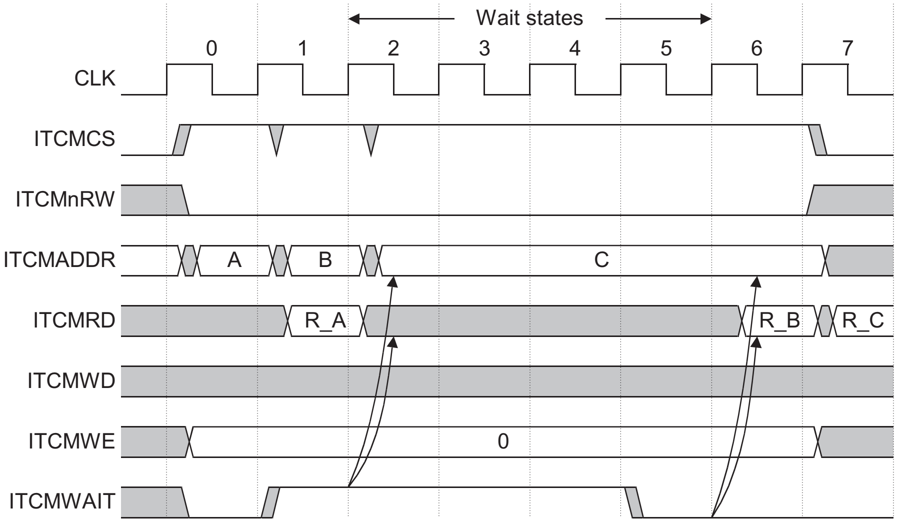


**Figure 6-4 TCM reads with four wait states**

### 6.6.4 TCM writes with zero wait states

Figure 6-5 is an example of single-cycle TCM write accesses. **ITCMWAIT** is never asserted, and there are no write delays. Write data must be driven in the same cycle as the address and the TCM control signals.

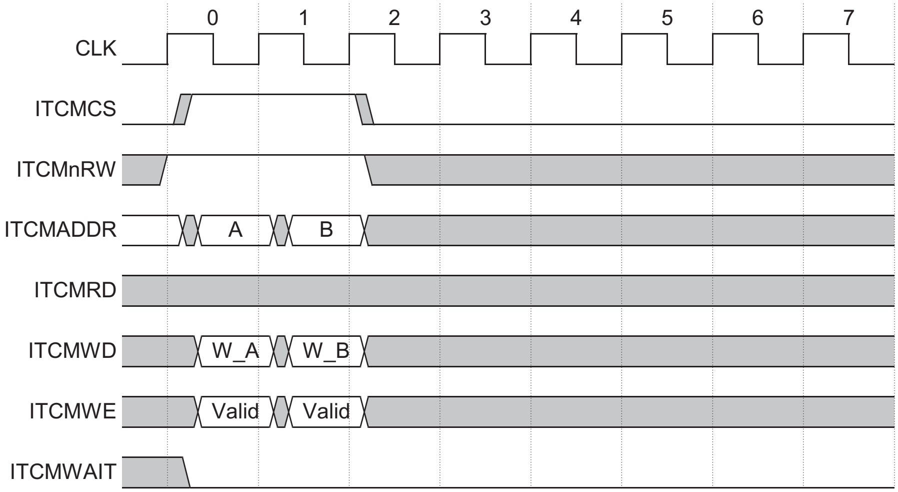


**Figure 6-5 TCM writes with zero wait states**

### 6.6.5 TCM write with one wait state

Figure 6-6 is an example of a two-cycle TCM write access. **ITCMWAIT** extends the completion of both the W_B and W_C writes for one cycle each. Write data must be driven in the same cycle as the address and the TCM control signals.

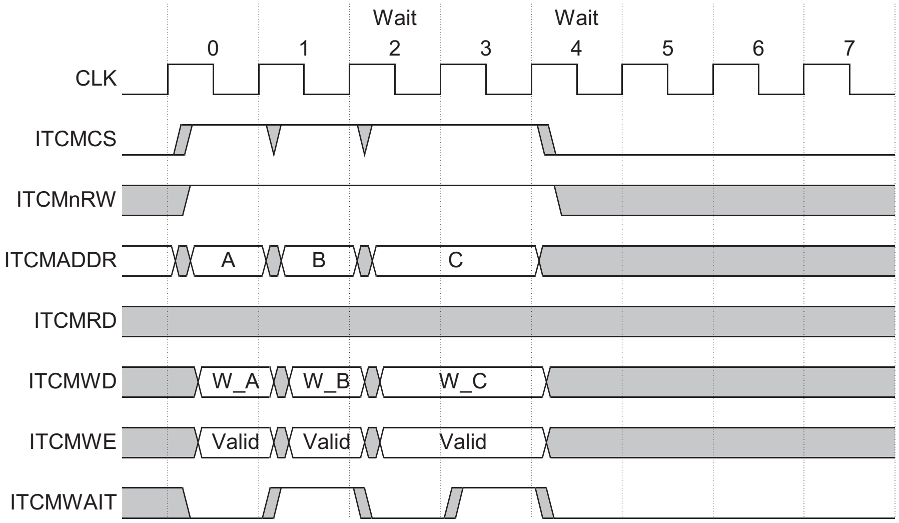


**Figure 6-6 TCM write with one wait state**

### 6.6.6 TCM write with two wait states

Figure 6-7 on page 6-14 is an example of a three-cycle TCM write access. **ITCMWAIT** extends the completion of both the W_B and W_C writes for two cycles each. Write data must be driven in the same cycle as the address and TCM control signals.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

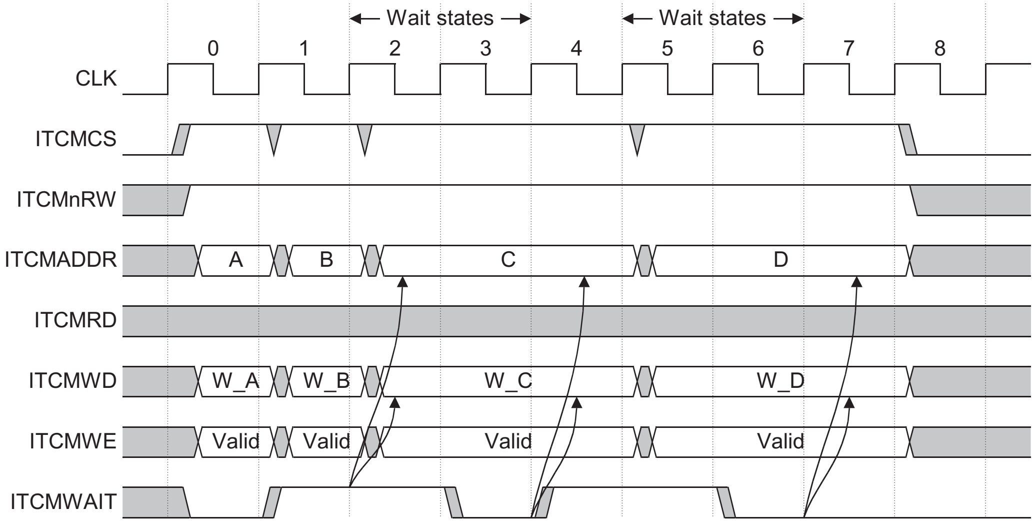


**Figure 6-7 TCM writes with two wait states**

### 6.6.7 TCM accesses with varying TCM wait states

Figure 6-8 shows a mix of read and write transfers with wait states of different lengths. The lengths of wait states are often transfer-dependent.

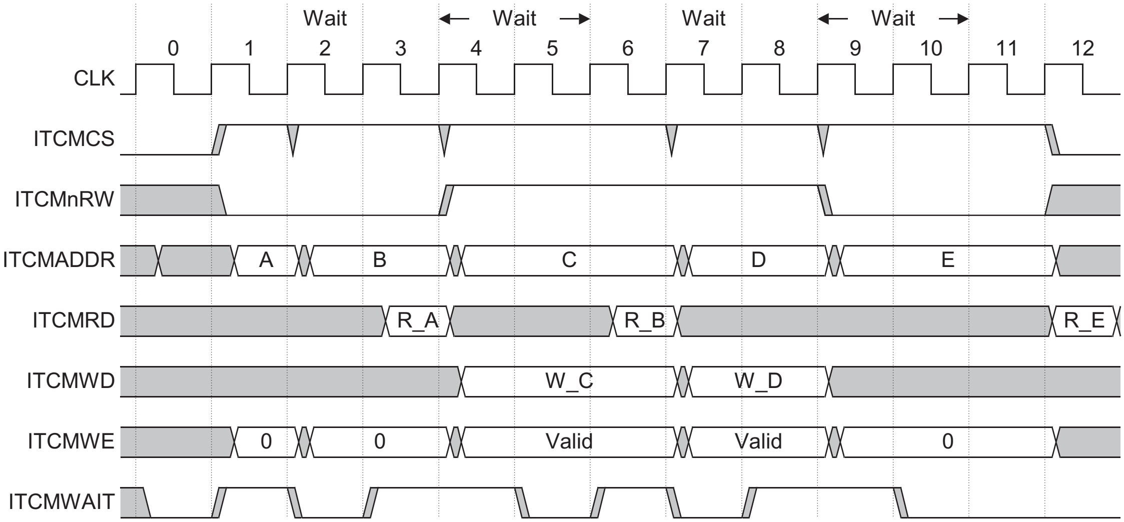


**Figure 6-8 TCM reads and writes with wait states of varying length**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 6.6.8 Speculative TCM read access

All read accesses to both the instruction and data TCMs are speculative. Because of the speculative nature of reads to both Instruction and Data TCMs, ARM Limited recommends that you do not use any read-sensitive memory or peripherals on the TCM ports. However, if the system must use read-sensitive memory, you can use a data buffer to hold the contents of the last read and forward that data in case of a consecutive read access to the same memory location.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 6.7 TCM implementation examples

This section contains the following examples:

- *Simplest zero-wait-state RAM example*
- *Byte-banks of RAM examples* on page 6-17
- *Multiple banks of RAM example* on page 6-18
- *Sequential RAM example* on page 6-19
- *Single or multiple wait-state RAM example* on page 6-20.

**Note**

The examples in this section are for the ITCM. They also apply to the D0TCM and D1TCM, except that each DTCM address range is **DxTCMADDR[21:3]**, half the address range of the ITCM.

The additional logic required for implementing the examples in this section is the responsibility of the implementer.

### 6.7.1 Simplest zero-wait-state RAM example

Figure 6-9 shows a single RAM device with a 32-bit data width connected directly to the TCM interface. The **ITCMWAIT** signal must be tied off to zero.

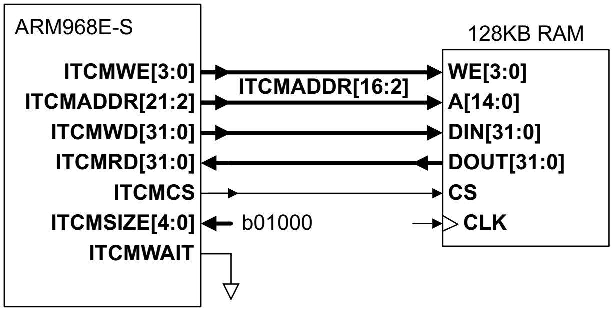


**Figure 6-9 Simplest zero wait state RAM example**

Inverters must be used if there are any polarity differences between the RAM input signals and those of the TCM interface. When the RAM chip select is active-LOW, an inverter must be placed between **ITCMCS** and the RAM chip select. This integration places a limit on the size of the TCMs. Multiple banks of RAM can be used to overcome this limitation. See *Multiple banks of RAM example* on page 6-18.

### 6.7.2 Byte-banks of RAM examples

If byte-write RAM is not available, you can use four banks of 8-bit wide RAM by routing each of the four bits of **ITCMWE** to one of the four RAM write enable inputs, as shown in Figure 6-10.

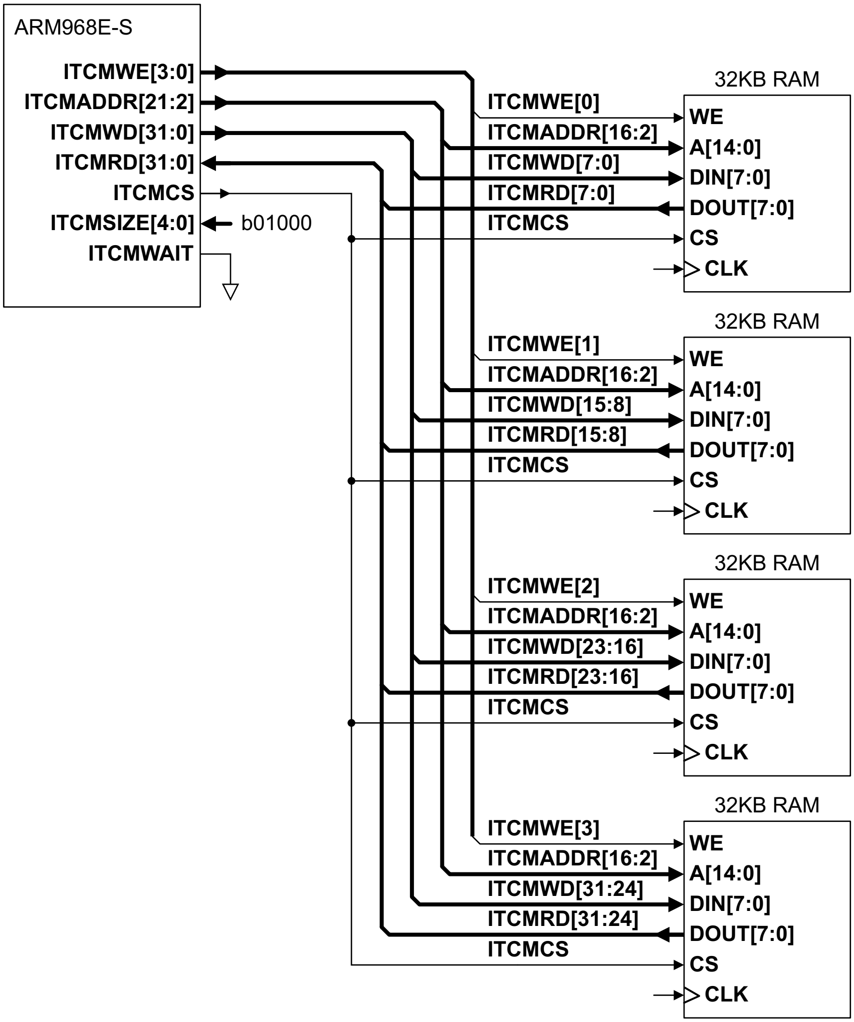


**Figure 6-10 Byte-banks of RAM example**

You can save a small amount of power by ANDing the chip select for each RAM device with **(ITCMWE** OR **ITCMnRW**). Then byte and halfword writes generate requests only to the required byte RAM as shown in Figure 6-11 on page 6-18.

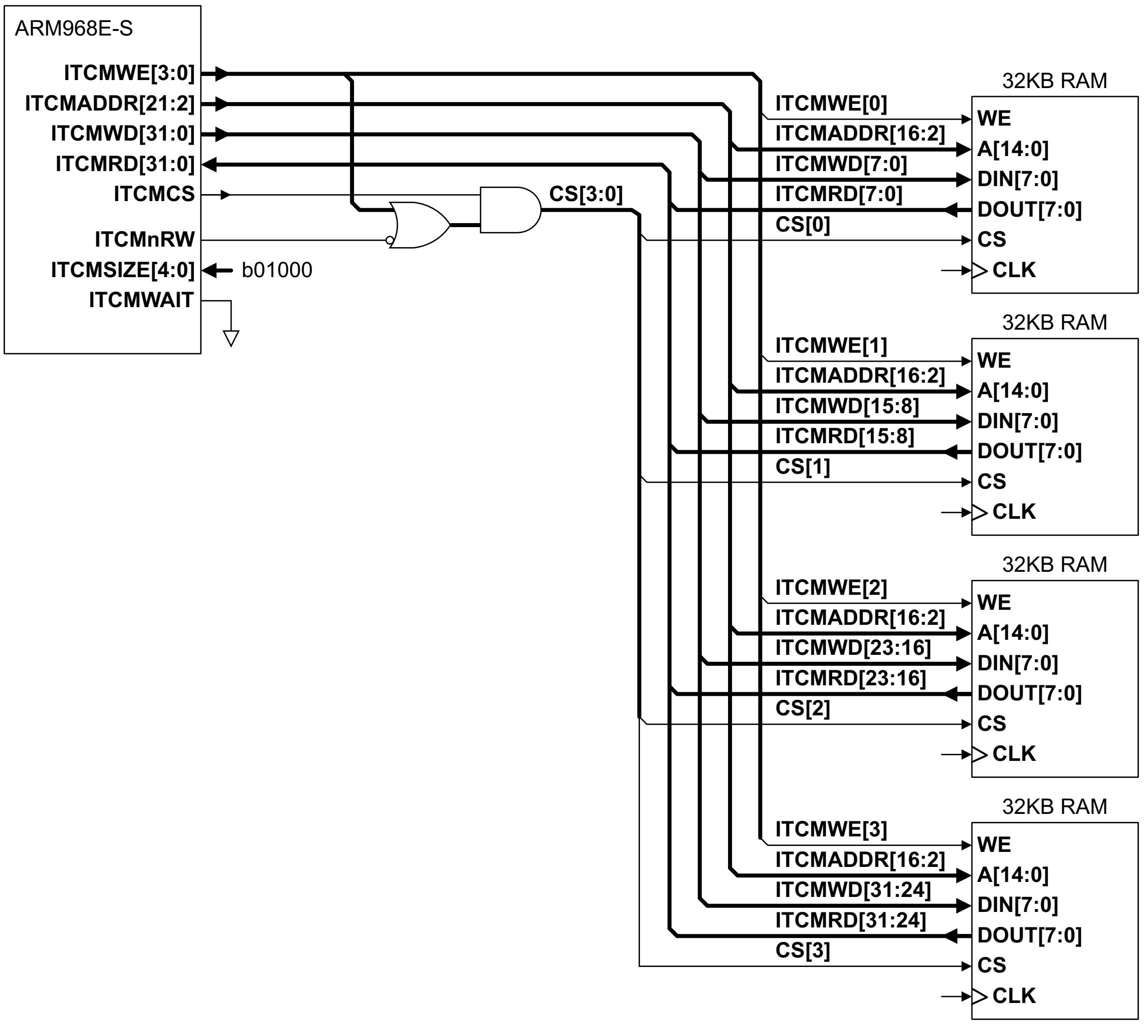


**Figure 6-11 Alternative byte-banks of RAM example**

### 6.7.3 Multiple banks of RAM example

With multiple RAM devices, the read data can come from one of two or more devices. To ensure that write data is not written to all devices, additional logic is required on either the chip select or the write enable. Figure 6-12 on page 6-19 shows an example of multiple banked RAMs with the chip select signal **ITCMCS** ANDed with the top address signal **ITCMADDR[18]**. This configuration is valid only if **ITCMWAIT** is tied LOW.

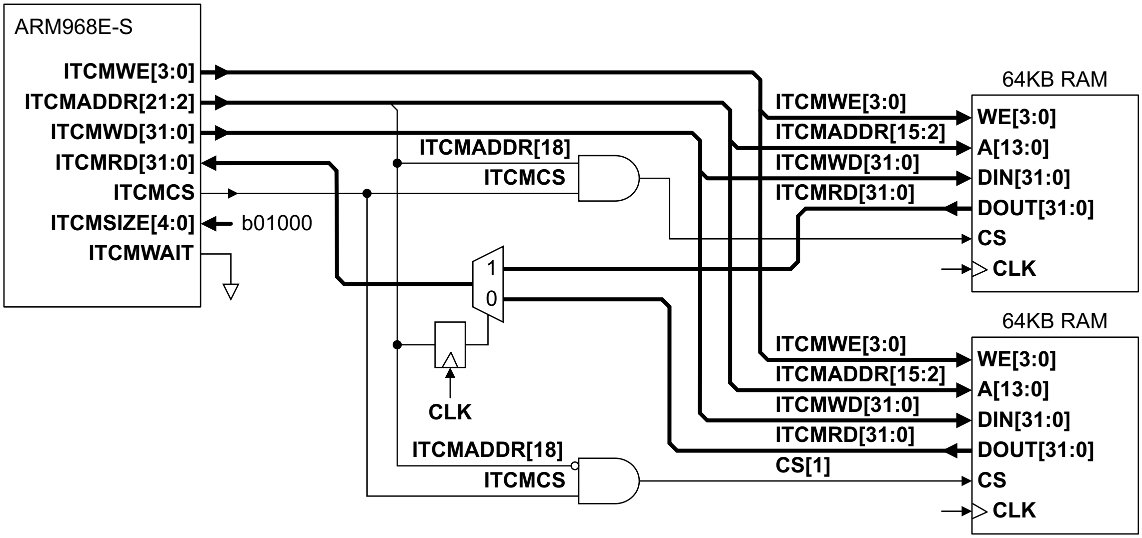


**Figure 6-12 Multiple banks of RAM example**

In the example shown in Figure 6-12, the reads and writes only occur on the RAM device required. This implementation uses less power than that shown in Figure 6-11 on page 6-18.

### 6.7.4 Sequential RAM example

If the RAM devices require a single wait-state except for sequential reads, the device can be connected as shown in Figure 6-13. The **ITCMWAIT** signal is derived from an inverter.

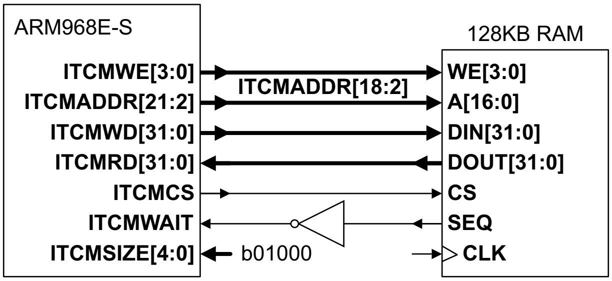


**Figure 6-13 Sequential RAM example**

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

### 6.7.5 Single or multiple wait-state RAM example

If the RAM devices require more than one cycle for all accesses, logic is required to assert the wait signal for the required number of cycles. Figure 6-14 shows a wait state controller asserting the wait signal as required. The power control block removes power to the TCM when it is not required.

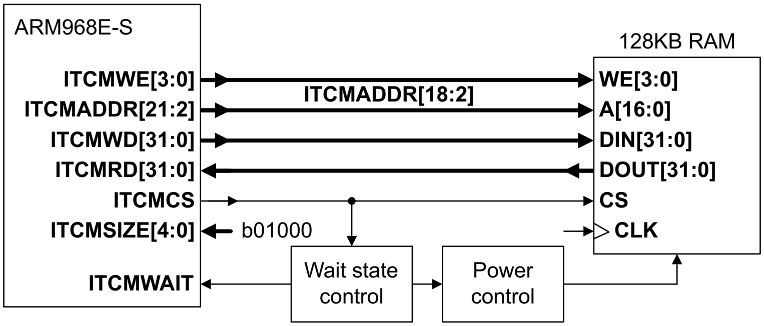


**Figure 6-14 Single or multiple wait-state RAM example**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 7
# DMA Interface

This chapter describes the interface for the *Direct Memory Access* (DMA) controller. It contains the following sections:

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 7.1 About the DMA interface

The ARM968E-S processor uses the *Advanced Microprocessor Bus Architecture* (AMBA) *Advanced High-performance Bus-Lite* (AHB-Lite) interface. The AHB-Lite version of the AMBA interface addresses the requirements of synthesizable high-performance designs, including:

- single rising-clock-edge operation
- unidirectional buses.

See the *AMBA Specification (Rev 2.0)* for full details of this bus architecture.

**Note**

The AHB-Lite architecture does not support RETRY or SPLIT responses from slaves.

The DMA interface implements the AHB-Lite bus slave interface. It is tightly integrated with the *Tightly-Coupled Memory* (TCM) interface to prevent access contention with the processor. The DMA clock enable, **HCLKEND**, enables transfer of data and code to and from the TCM even while the processor is in the low-power wait-for-interrupt state. **HCLKEND** also controls the frequency ratio of **CLK** to the DMA controller bus.

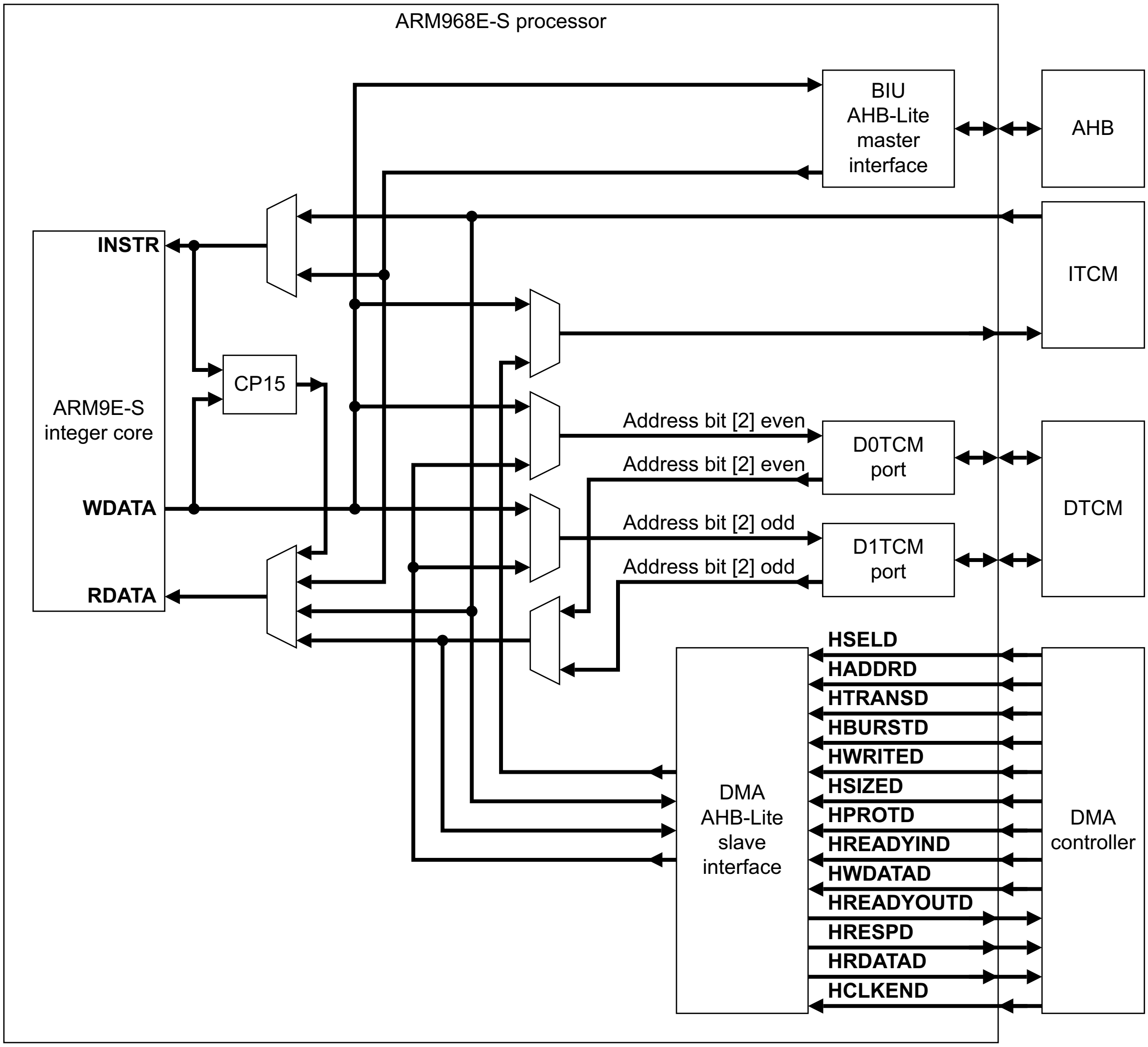


**Figure 7-1 DMA interface**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 7.2 Bus transfer characteristics

The AHB-Lite DMA interface handles all data transfers between an external DMA controller and the TCM in a way that is transparent to the processor. The types of DMA interface transfers are:

- data writes
- data reads.

Each of the AMBA AHB bus transfers generates a signature. Table 7-1 lists the types of DMA interface transfers and their characteristics.

**Table 7-1 DMA transfer characteristics**

| Transfer | HADDRD | HTRANSD | HSIZED | HWRITED | HRDATAD/<br>HWDATAD | BIGEND |
|---|---|---|---|---|---|---|
| Data write | [31:2] b00 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [7:0] | 0 |
| Data write | [31:2] b01 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [15:8] | 0 |
| Data write | [31:2] b10 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [23:16] | 0 |
| Data write | [31:2] b11 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [31:24} | 0 |
| Data write | [31:2] b0x | NS- . . . -NS or NS-S- . . . -S | Halfword | 1 | [15:0] | 0 |
| Data write | [31:2] b1x | NS- . . . -NS or NS-S- . . . -S | Halfword | 1 | [31:16] | 0 |
| Data write | [31:2] b00 | NS- . . . -NS or NS-S- . . . -S | Word | 1 | [31:0] | 0 |
| Data write | [31:2] b00 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [31:24] | 1 |
| Data write | [31:2] b01 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [23:16] | 1 |
| Data write | [31:2] b10 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [15:8] | 1 |
| Data write | [31:2] b11 | NS- . . . -NS or NS-S- . . . -S | Byte | 1 | [7:0] | 1 |
| Data write | [31:2] b0x | NS- . . . -NS or NS-S- . . . -S | Halfword | 1 | [31:16] | 1 |
| Data write | [31:2] b1x | NS- . . . -NS or NS-S- . . . -S | Halfword | 1 | [15:0] | 1 |
| Data write | [31:2] b00 | NS- . . . -NS or NS-S- . . . -S | Word | 1 | [31:0] | 1 |
| Data read | [31:0] | NS- . . . -NS or NS-S- . . . -S | Word | 0 | [31:0] | - |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 7.2.1 Transfer size

**HSIZED[1:0]**, **HADDRD[1:0]**, and **BIGEND** define the transfer size of a write transfer.

Table 7-2 shows how the transfer size (**HSIZED[1:0]**), address offset (**HADDRD[1:0]**), and endian format (**BIGEND**) determine which byte lanes are used in big-endian writes.

**Table 7-2 Active byte lanes with a 32-bit big-endian data bus**

| Transfer size | Address offset | HWDATAD [31:24] | HWDATAD [23:16] | HWDATAD [15:8] | HWDATAD [7:0] |
|---|---|---|---|---|---|
| Word | 0 | Active | Active | Active | Active |
| Halfword | 0 | Active | Active | - | - |
| Halfword | 2 | - | - | Active | Active |
| Byte | 0 | Active | - | - | - |
| Byte | 1 | - | Active | - | - |
| Byte | 2 | - | - | Active | - |
| Byte | 3 | - | - | - | Active |

Table 7-3 shows how the transfer size (**HSIZED[1:0]**), address offset (**HADDRD[1:0]**), and endian format (**BIGEND**) determine which byte lanes are used in little-endian writes.

**Table 7-3 Active byte lanes with a 32-bit big-endian data bus**

| Transfer size | Address offset | HWDATAD [31:24] | HWDATAD [23:16] | HWDATAD [15:8] | HWDATAD [7:0] |
|---|---|---|---|---|---|
| Word | 0 | Active | Active | Active | Active |
| Halfword | 0 | - | - | Active | Active |
| Halfword | 2 | Active | Active | - | - |
| Byte | 0 | - | - | - | Active |
| Byte | 1 | - | - | Active | - |
| Byte | 2 | - | Active | - | - |
| Byte | 3 | Active | - | - | - |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 7.2.2 Sequential and nonsequential transfers

The HTRANSD column in Table 7-1 on page 7-4 shows that transfers on the DMA interface can be sequential (NS- S- . . . -S) or nonsequential (NS- . . . -NS).

### 7.2.3 Burst types

The DMA interface supports all burst types.

### 7.2.4 Protection control

The **HPROTD[3:0]** signals are not implemented and have no effect on the operation of the DMA interface.

### 7.2.5 Error response limitations

The DMA interface supports error responses from the TCM for data reads but ignores error responses for write transfers.

If the TCM error detection logic generates an error response during a DMA read, the DMA interface drives the **HRESPD** signal HIGH. It is the responsibility of the software to disregard that data transfer and request the data again.

## 7.3 AHB bus slave interface

The DMA interface supports the AHB-Lite bus slave interface. See the *AMBA Specification* (Rev 2.0) for a detailed description of the AHB protocol.

The AHB architecture is based on separate cycles for address and data. The address and control values for an access are broadcast from the rising edge of **CLK** in the cycle before the data is expected to be read or written. During this data cycle, the address and control values for the next cycle are driven out, providing fully-pipelined bus operation.

When an access is in its data cycle, the DMA interface can wait the access by driving the **HREADYD** response LOW. This has the effect of stretching the current data cycle and the pipelined address and control for the next access. This creates a system in which the DMA controller, the AHB master, samples **HREADYD** on the rising edge of **CLK** to determine if an access is complete and a new address can be sampled or driven out.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 7.4 Wait-for-interrupt mode

The wait-for-interrupt instructions put the ARM968E-S processor into a low-power state:

```
MCR p15, 0, Rd, c7, c0, 4
```

```
MCR p15, 0, Rd, c15, c8, 2
```

Either of these instructions stops the internal processor clocks until an interrupt (IRQ or FIQ) or a debug request (**EDBGRQ**) occurs. The switch into wait-for-interrupt mode is delayed until all write buffers are drained and the memory system is in a quiescent state.

The DMA interface has a separate clock enable, **HCLKEND**, that enables the DMA and TCM interfaces to continue operating while the ARM968E-S processor is in wait-for-interrupt mode. This feature enables you to use the external DMA controller to transfer data to and from the TCM before waking the processor.

Entering wait-for-interrupt mode asserts the **STANDBYWFI** signal. The system designer can use **STANDBYWFI** to shut down clocks to other SoC blocks that do not have to be clocked when the ARM968E-S processor is idle.

The **STANDBYWFI** signal is deasserted in the cycle following an interrupt or a debug request. It is guaranteed that no form of access on any external interface is started until the cycle after **STANDBYWFI** is deasserted. Figure 7-2 shows the deassertion of the **STANDBYWFI** signal after an IRQ interrupt.

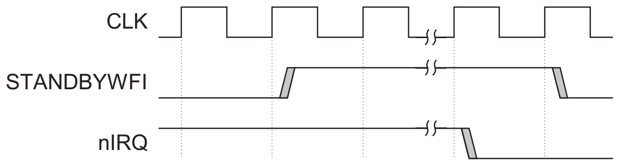


**Figure 7-2 Deassertion of STANDBYWFI after an IRQ interrupt**

When the processor enters a low-power state, all of the main internal clocks are stopped. However, the processor is active if **DBGTCKEN** is asserted.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 7.5 AHB transfer descriptions

The DMA interface performs a subset of the possible AHB bus transfers. This section describes the transfers that can be performed.

### 7.5.1 DMA reads

Figure 7-3 shows the bus activity for a DMA read from the *Instruction TCM* (ITCM). Because the address and control data have to be registered to meet the TCM setup time, a DMA read always has a latency of at least one AHB cycle.

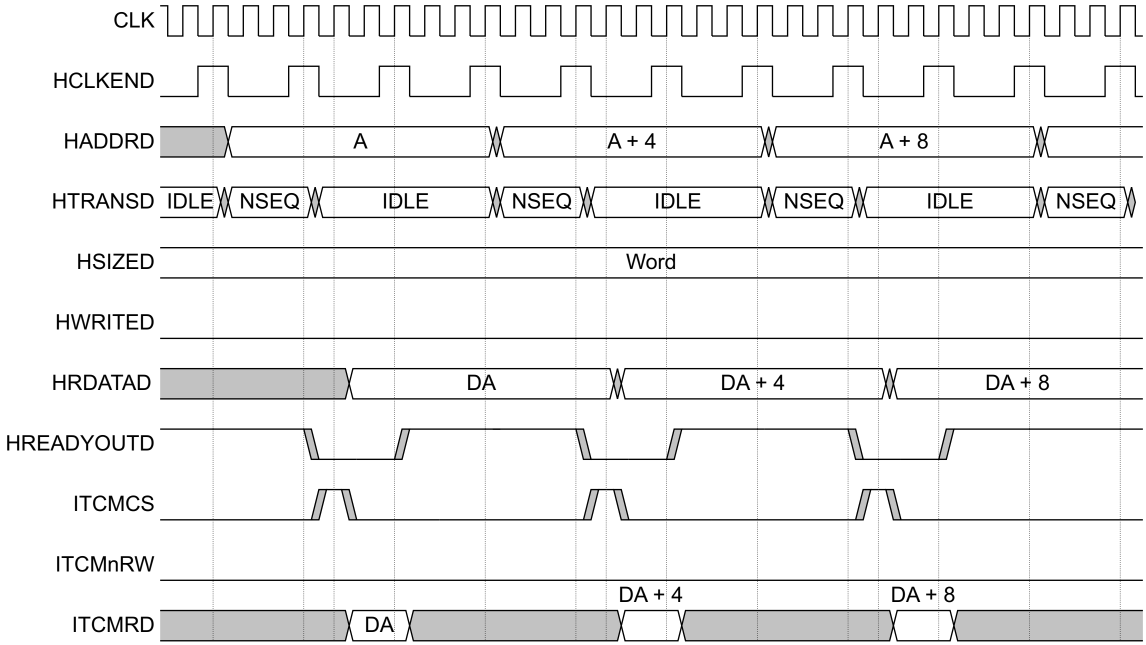


**Figure 7-3 DMA reads of ITCM**

### 7.5.2 DMA read with error response

Figure 7-4 on page 7-10 shows an example of a DMA read to the ITCM with the ITCM generating an error by asserting the **ITCMERROR** signal. As soon as the DMA interface samples the error, it asserts the **HRESPD** signal on the next valid AHB cycle, that is, when **HCLKEND** is HIGH. This is to indicate to the DMA controller that there was an error on the read transaction. An error always requires a two-cycle response.

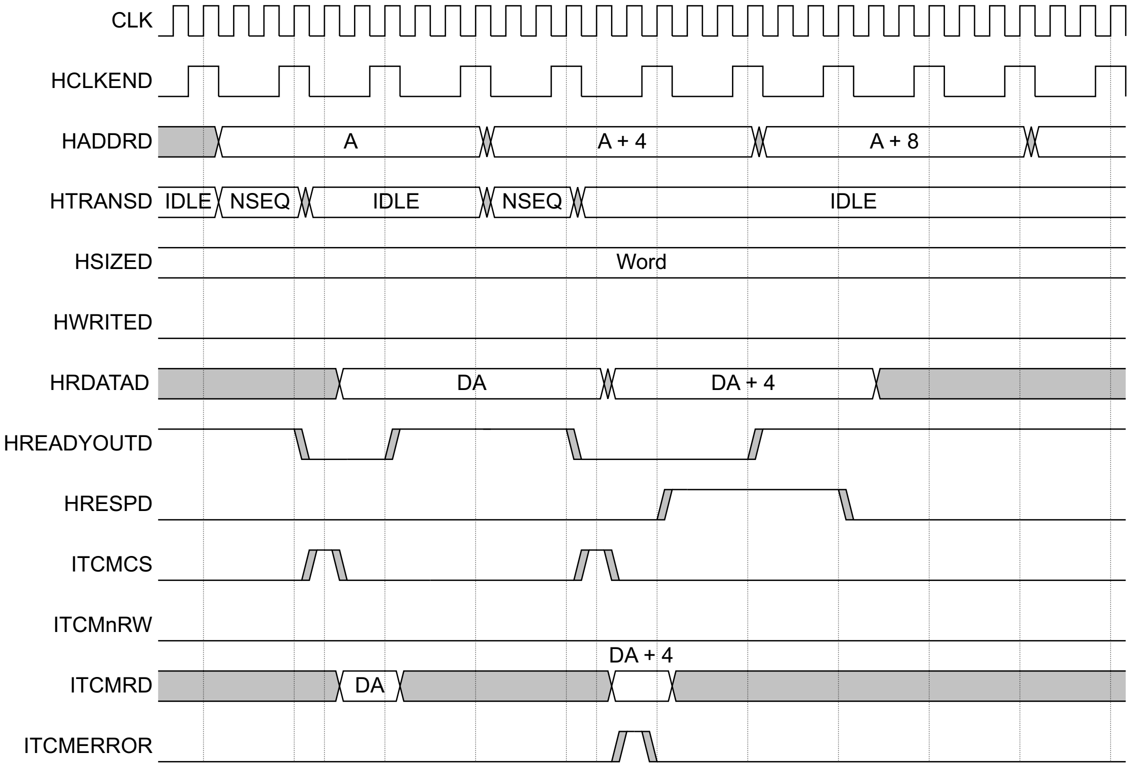


**Figure 7-4 DMA read of ITCM with error response**

### 7.5.3 DMA read with wait state

Figure 7-5 on page 7-11 shows a DMA read transaction with a wait state. In the data phase, the TCM generates a wait state that the DMA interface then samples. As soon as the DMA interface recognizes that the TCM is unable to fulfill the transaction, it forces **HREADYOUTD** LOW to inform the DMA controller to extend the data portion of the AHB transfer.

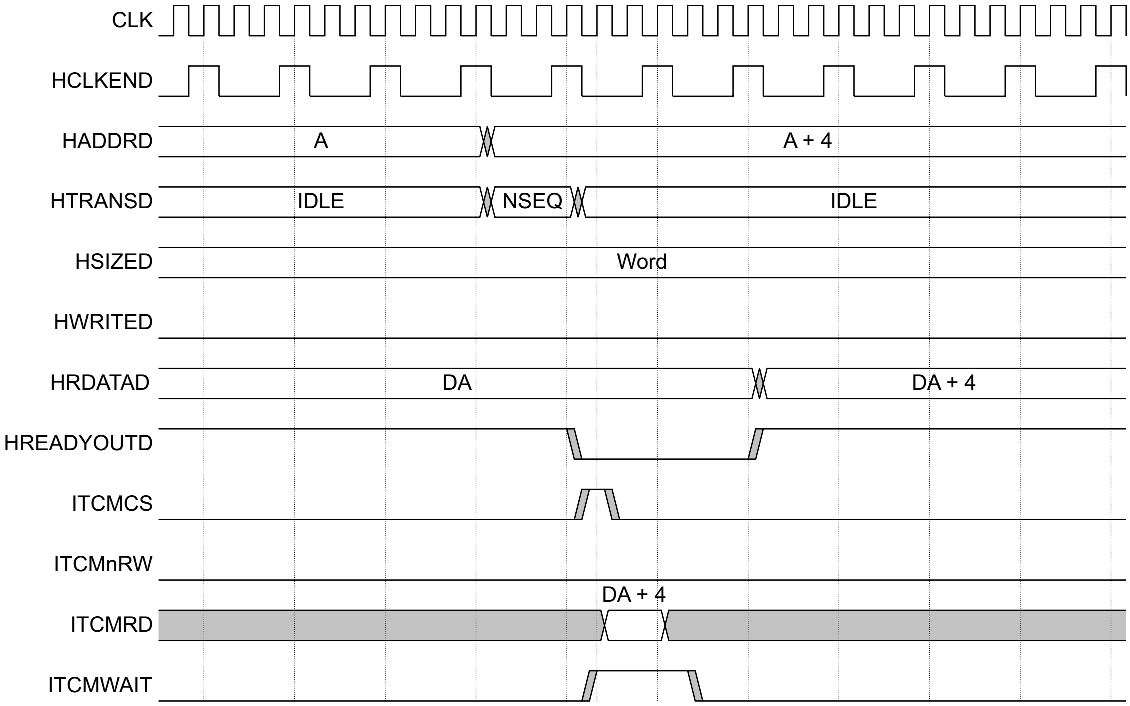


**Figure 7-5 DMA read of ITCM with wait state**

### 7.5.4 DMA write with wait state

Figure 7-6 on page 7-12 is similar to Figure 7-5 except that it shows a DMA write operation instead of a read. The effect of asserting the **D0TCMWAIT** or **D1TCMWAIT** signal causes the data phase to be extended.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

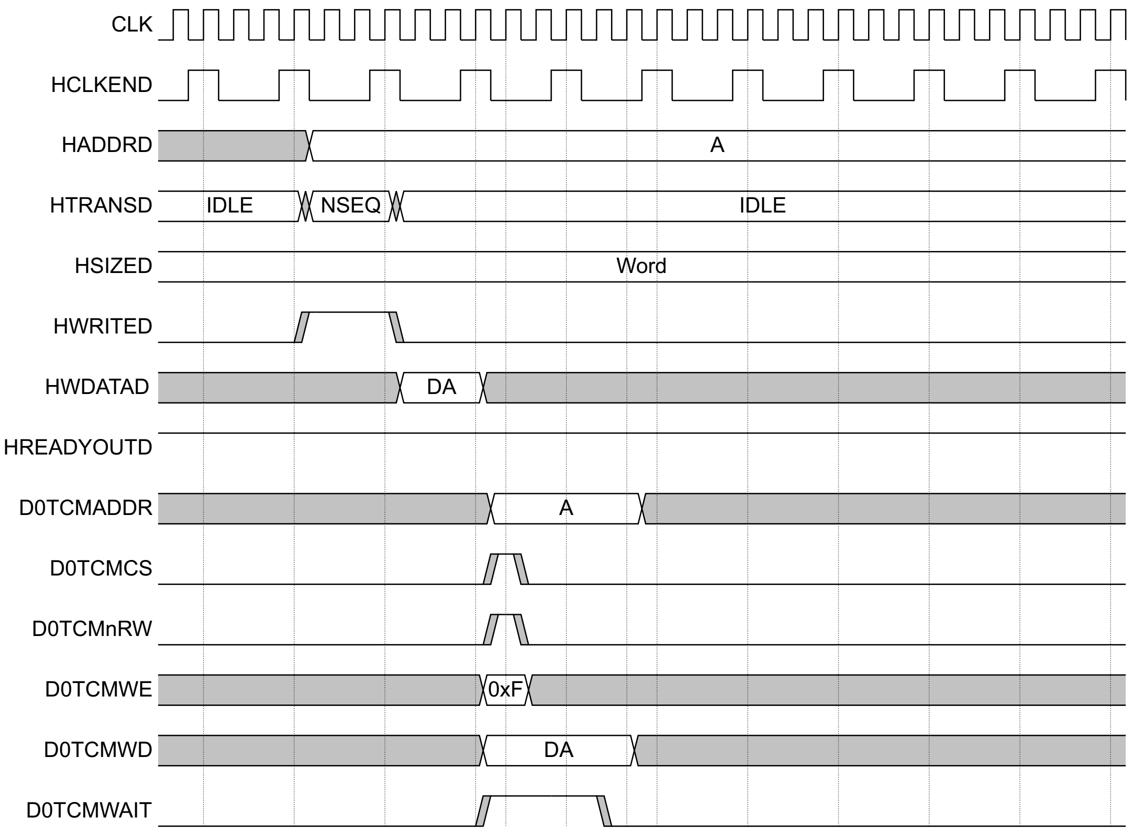


**Figure 7-6 DMA write of DTCM with wait state**

### 7.5.5 Interleaved DMA writes to DTCM

Figure 7-7 on page 7-13 shows the interleaved bus activity when the DMA interface is writing a block of data to the DTCM. You can see how the D0TCM and D1TCM ports use alternate cycles for each word as the transactions progress.

**Note**

The ARM968E-S processor does not support interleaved access of the ITCM.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

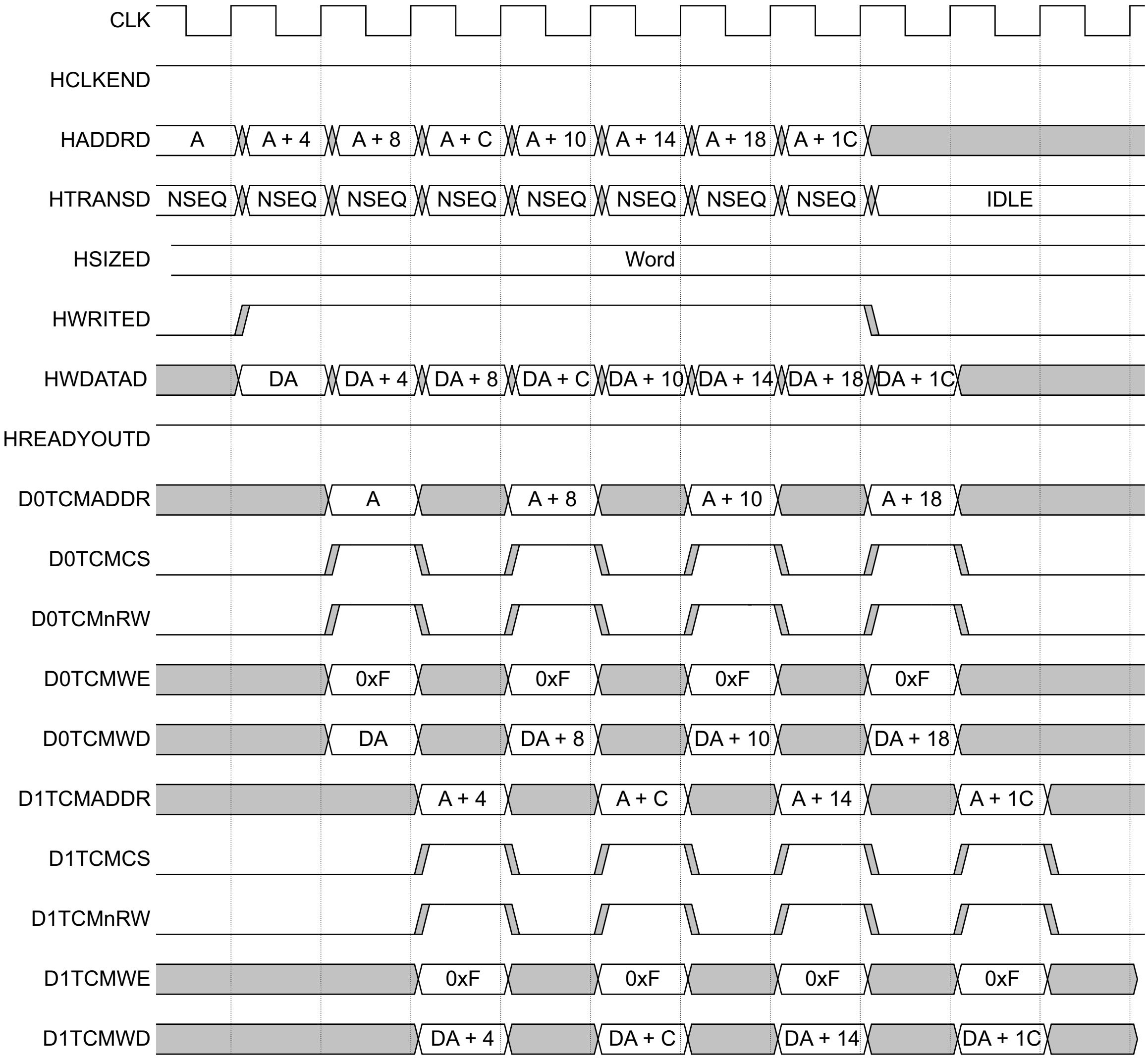


**Figure 7-7 Interleaved DMA writes to DTCM**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 8
# Debug Support

This chapter describes the ARM968E-S debug interface. It contains the following sections:

- *About the debug interface* on page 8-2
- *Debug systems* on page 8-4
- *Debug data chain 15* on page 8-6
- *Debug interface signals* on page 8-8
- *ARM9E-S core clock domains* on page 8-13
- *Determining the core and system state* on page 8-14.

This chapter also describes the ARM9E-S EmbeddedICE-RT logic in the following sections:

- *About the EmbeddedICE-RT* on page 8-15
- *Disabling EmbeddedICE-RT* on page 8-17
- *The debug comms channel* on page 8-18
- *Monitor debug-mode* on page 8-22
- *Additional debug reading* on page 8-24.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.1 About the debug interface

The ARM968E-S debug interface is based on IEEE Std. 1149.1-1990, Standard Test Access Port and Boundary-Scan Architecture. See this standard for an explanation of the terms used in this chapter and for a description of the TAP controller states.

The ARM968E-S processor contains hardware extensions for advanced debugging features to facilitate application software and operating system development.

The debug extensions enable you to force the processor into debug state. Debug state effectively stops the ARM968E-S processor and memory system and isolates them from the rest of the system. This is known as halt mode operation. It enables you to examine the internal state of the processor and the external state of the AHB while all other system activity continues as normal. When debug is complete, the processor restores the system state and resumes program execution.

In addition, the processor supports a real-time debug mode that generates an internal Instruction Abort or Data Abort instead of a breakpoint or watchpoint. This is known as monitor debug-mode operation.

When using a debug monitor program activated by the Abort exception, you can debug the ARM968E-S processor while critical interrupt service routines are executing. The debug monitor program typically communicates with the debug host over the debug comms channel. See *Monitor debug-mode* on page 8-22.

In debug state, you can examine the internal state of the ARM9E-S core by serially inserting instructions into the instruction pipeline without using the external data bus. For example, you can insert a store multiple (STM) instruction and shift out the contents of the ARM9E-S registers without affecting the rest of the system.

### 8.1.1 Entering debug state

A request on one of the external debug interface signals or on an internal functional unit known as the EmbeddedICE-RT logic forces the ARM9E-S core into debug state. The interrupts that activate debug are:

- a breakpoint (a given instruction fetch)
- a watchpoint (a data access)
- an external debug request.

### 8.1.2 Clocks

The system and test clocks must be synchronized externally to the processor. The Multi-ICE debug agent directly supports one or more cores within an ASIC design. Synchronizing off-chip debug clocking with the processor requires a three-stage synchronizer. An off-chip device such as Multi-ICE issues a **TCK** signal and waits for

Copyright © 2004, 2006 ARM Limited. All rights reserved.

the *Returned TCK* (**RTCK**) signal to come back. The off-chip device maintains synchronization because it does not progress to the next **TCK** until after **RTCK** is received.

Figure 8-1 shows this synchronization.

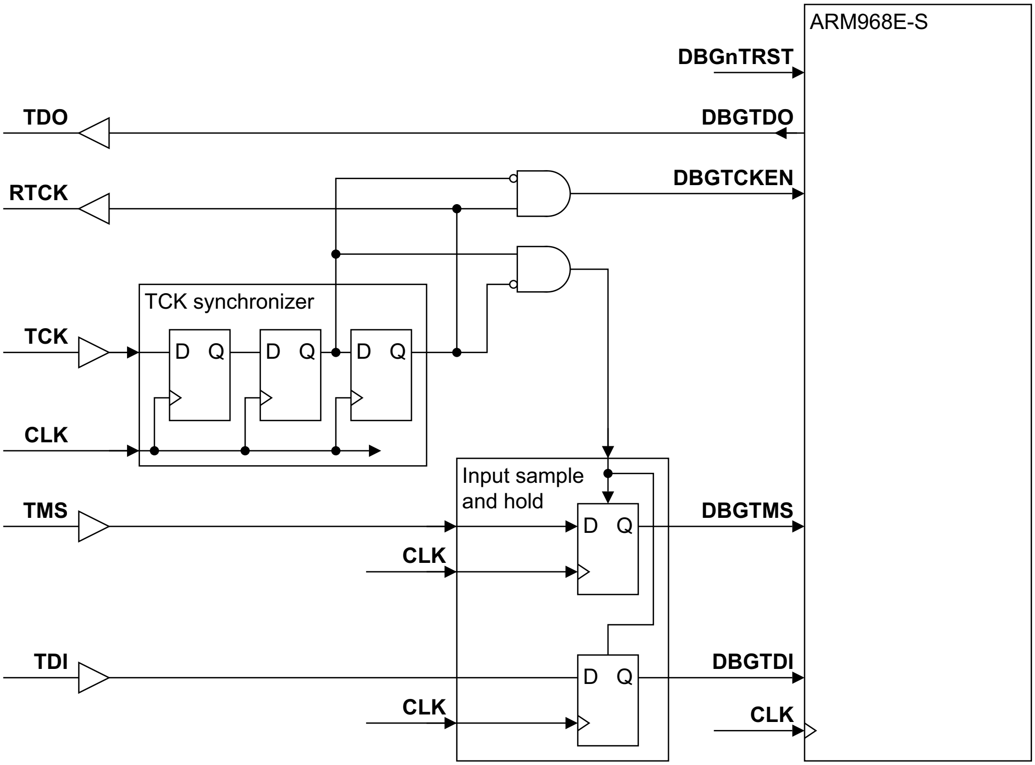


**Figure 8-1 Clock synchronization**

## 8.2 Debug systems

The ARM968E-S processor forms one component of a debug system that interfaces from the high-level debugging performed by you to the low-level interface supported by the processor. Figure 8-2 shows a typical debug system.

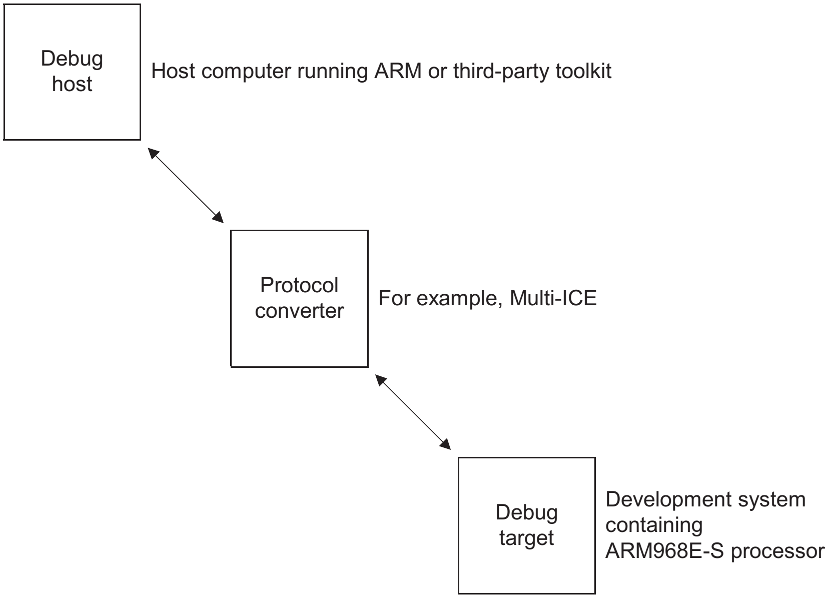

Debug host
Host computer running ARM or third-party toolkit
Protocol converter
For example, Multi-ICE
Debug target
Development system containing ARM968E-S processor

**Figure 8-2 Typical debug system**

A debug system has three parts:

- *Debug host*
- *Protocol converter* on page 8-5
- *ARM968E-S debug target* on page 8-5.

The debug host and the protocol converter are system-dependent.

### 8.2.1 Debug host

The debug host is the computer that runs a software debugger, such as armsd. Through the debug host, you can issue high-level commands such as setting breakpoints or examining the contents of memory.

### 8.2.2 Protocol converter

An interface, such as a parallel port, connects the debug host to the ARM968E-S development system. The protocol converter converts the messages broadcast over this connection to the interface signals of the ARM968E-S processor.

### 8.2.3 ARM968E-S debug target

The ARM968E-S processor has hardware extensions to facilitate debugging at the lowest level. The debug extensions enable you to:

- stall program execution
- examine the internal state of the core
- examine the state of the memory system
- resume program execution.

The following major blocks of the ARM9E-S debug model are shown in Figure 8-3.

**ARM9E-S core**

This includes hardware support for debug.

**EmbeddedICE-RT logic**

This is a set of registers and comparators used to generate debug exceptions such as breakpoints. This unit is described in *About the EmbeddedICE-RT* on page 8-15.

**TAP controller** This controls the action of the debug data chains using a JTAG serial interface.

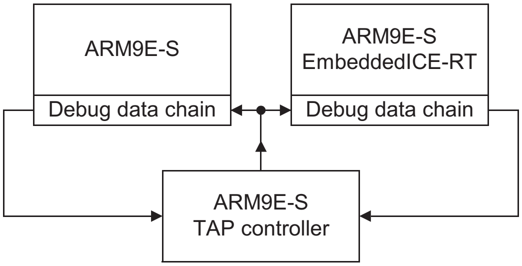


**Figure 8-3 Block diagram of the ARM9E-S debug model**

## 8.3 Debug data chain 15

Debug data chain 15 enables:

- debug access to the CP15 register bank
- configuring the processor state while in debug state.

The order of debug data chain 15 from the **DBGTDI** input to the **DBGTDO** output is shown in Table 8-1.

**Table 8-1 Debug data chain 15 bit order**

| Bits | Contents |
|---|---|
| [38] | 1 = write<br>0 = read. |
| [37:32] | CP15 register address |
| [31:0] | CP15 register value |

The CP15 register address field of debug data chain 15 provides debug access to the CP15 registers as shown in Table 8-2.

**Table 8-2 Mapping of debug data chain 15 address field to CP15 registers**

| Bit [38] | Bits [37:32] | Bits [31:30] | CP15 register number | Meaning |
|---|---|---|---|---|
| b0 | b0 0000 0 | bxx | c0 | Read ID register |
| b0 | b0 0001 0 | bxx | c1 | Read control register |
| b1 | b0 0001 0 | bxx | c1 | Write control register |

The scan address decode overloads the existing functional decode logic that accesses the CP15 registers during MCR and MRC instructions. See *CP15 register descriptions* on page 4-5.

The decode overload is performed as follows:

**Bit [37]** This bit corresponds to opcode_1 of an MCR or MRC instruction.

**Bits [36:33]** These bits correspond to CRn of an MCR or MRC instruction.

**Bit [32]** This bit corresponds to bit 0 of opcode_2 of an MCR or MRC instruction.

**Bits [2:1]** Bits [2:1] of opcode_2 are tied to b00 during debug state.

Debug data chain 15 provides access to only bit 0 of the opcode_2 field by default.

The ability to control the ARM968E-S system state through debug data chain 15 provides extra debug visibility. For example, to compare the contents of an address that maps to the ITCM or DTCM with the same address in AHB memory, the debugger can:

1. Load from the address with the TCM enabled to return the TCM data.
2. Disable the TCM.
3. Perform the load again. Because the TCM is disabled, the second load accesses the AHB and returns the value from AHB memory.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.4 Debug interface signals

There are four primary external signals in the full ARM968E-S debug interface:

- **DBGIEBKPT**, **DBGDEWPT**, and **EDBGRQ** are system requests for the processor to enter debug state
- **DBGACK** is used by the processor to flag back to the system that it is in debug state.

### 8.4.1 Entry into debug state on breakpoint

Any instruction being fetched from memory is sampled at the end of a cycle. To apply a breakpoint to an instruction, the breakpoint signal must be asserted by the end of the same cycle. This is shown in Figure 8-4 on page 8-9.

You can build external logic, such as additional breakpoint comparators, to extend the breakpoint functionality of the EmbeddedICE-RT logic. These outputs must be applied to the **DBGIEBKPT** input. This signal is ORed with the internally-generated breakpoint signal before being applied to the ARM9E-S core control logic. The timing of the input makes it unlikely that data-dependent external breakpoints are possible.

A breakpointed instruction can enter the Execute stage of the pipeline, but any state change as a result of the instruction is prevented. All writes from previous instructions complete as normal.

The Decode cycle of the debug entry sequence occurs during the Execute cycle of the breakpointed instruction. The latched breakpoint signal forces the processor to start the debug sequence.

**Note**

The ARM968E-S processor performs Thumb instruction fetches as 32-bit accesses to the AHB or TCM interfaces. As a result, external breakpoint hardware cannot identify which halfword has been requested by the ARM9E-S core as an instruction. If an external hardware breakpoint detector generates an external breakpoint, it applies to both instructions in the 32-bit word fetched from memory. External breakpoints in Thumb state must be avoided as program execution might be interrupted unintentionally. To ensure precise debug entry, use the Embedded-ICE module within the ARM9E-S core.

Figure 8-4 on page 8-9 shows breakpoint timing.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

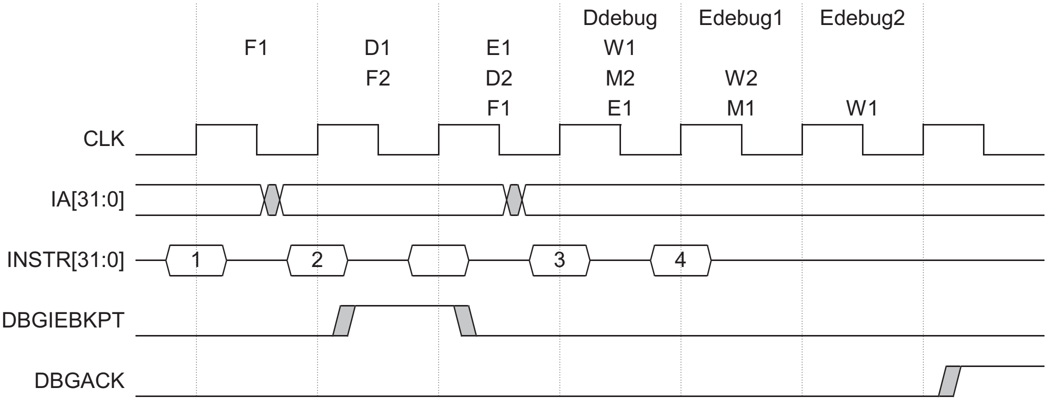


**Figure 8-4 Breakpoint timing**

### 8.4.2 Breakpoints and exceptions

If a breakpointed instruction has a Prefetch Abort associated with it, the Prefetch Abort takes priority, and the breakpoint is ignored.

SWI and Undefined instructions are treated in the same way as any other instruction that might have a breakpoint set on it. Therefore, the breakpoint takes priority over the SWI or Undefined instruction.

If there is a breakpointed instruction on an instruction boundary and an **nIRQ** or **nFIQ** interrupt, the interrupt is taken and the breakpointed instruction is discarded. When the interrupt is being serviced, the execution flow returns to the original program. The instruction that was previously breakpointed is fetched again, and if the breakpoint is still set, the processor enters debug state when the instruction reaches the Execute stage of the pipeline.

After the processor enters halt mode debug state, it is important that interrupts do not affect the instructions executed. Entering halt mode debug state disables interrupts but does not affect the state of the I and F bits in the *Program Status Register* (PSR).

### 8.4.3 Watchpoints

Because of the nature of the pipeline, entry into debug state following a watchpointed memory access is imprecise.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

External logic, such as external watchpoint comparators, can extend the functionality of the EmbeddedICE-RT logic. Their output must be applied to the **DBGDEWPT** input. This signal is ORed with the internally-generated **Watchpoint** signal before being applied to the ARM9E-S core control logic. The timing of the input makes it unlikely that data-dependent external watchpoints are possible.

After a watchpointed access, the next instruction in the processor pipeline completes its execution. When this instruction is a single-cycle data-processing instruction, entry into debug state is delayed for one cycle while the instruction completes. The timing of debug entry following a watchpointed load in this case is shown in Figure 8-5.

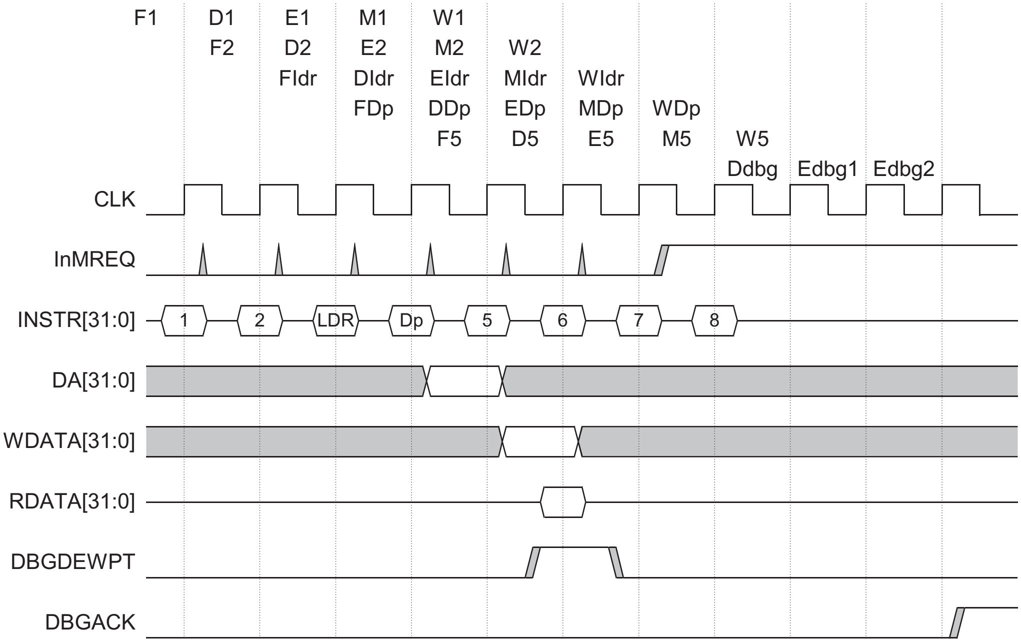


**Figure 8-5 Watchpoint entry with data processing instruction**

**Note**

Although instruction 5 enters the Execute stage, it is not executed, and there is no state update as a result of this instruction. When the debugging session is complete, normal continuation involves a return to instruction 5, the next instruction in the code sequence to be executed.

The instruction following the instruction that generated the watchpoint might have modified the *Program Counter (PC)*. If this happens, it is not possible to determine the instruction that caused the watchpoint. A timing diagram showing debug entry after a watchpoint where the next instruction is a branch is shown in Figure 8-6. However, it is always possible to restart the processor.

When the processor enters debug state, the ARM9E-S core is interrogated to determine its state. In the case of a watchpoint, the PC contains a value that is five instructions on from the address of the next instruction to be executed. Therefore, if on entry to debug state, in ARM state, the instruction `SUB PC, PC, #20` is scanned in and the processor restarted, execution flow returns to the next instruction in the code sequence.

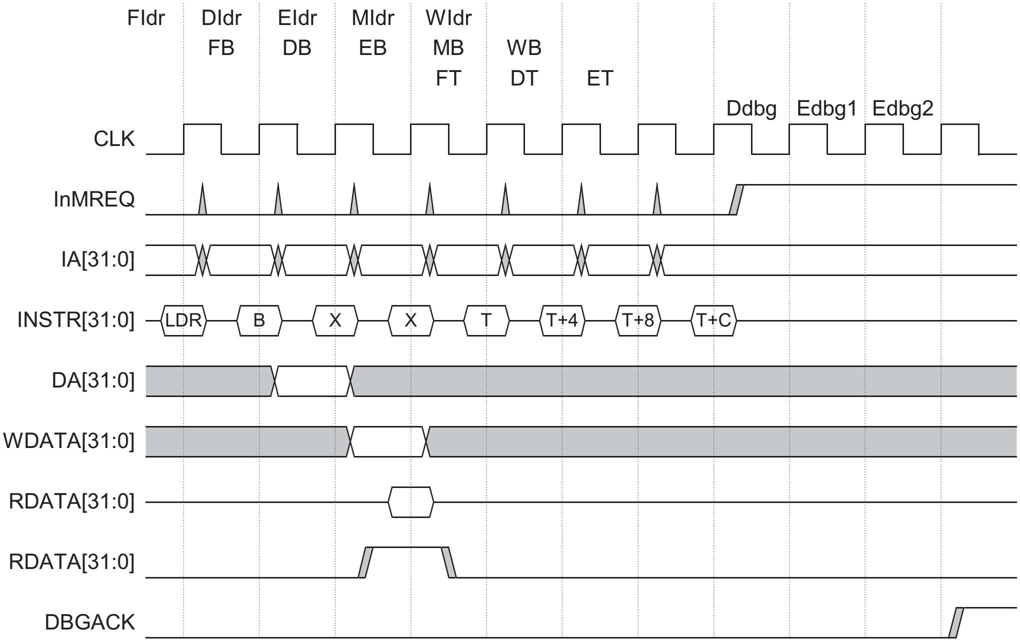


**Figure 8-6 Watchpoint entry with branch**

### 8.4.4 Watchpoints and exceptions

If the watchpointed data access aborts, the processor:

- latches the watchpoint condition
- performs the exception entry sequence
- enters debug state.

If there is an interrupt pending, the ARM9E-S core completes the exception entry sequence and then enters debug state.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### 8.4.5 Debug request

A debug request can take place through the EmbeddedICE-RT logic or by asserting the **EDBGRQ** signal. The request is synchronized and passed to the processor. Debug request takes priority over any pending interrupt. Following synchronization, the core enters debug state when the instruction at the Execute stage of the pipeline is completed (when Memory and Write stages of the pipeline have completed). While waiting for the instruction to finish executing, no more instructions are issued to the Execute stage of the pipeline.

**Caution**

Asserting **EDBGRQ** in monitor debug-mode results in Unpredictable behavior.

### 8.4.6 Actions of the ARM9E-S core in debug state

When the ARM9E-S core is in debug state, both memory interfaces indicate internal cycles. This ensures that both the tightly-coupled memory and the AHB interface are quiescent, and that the rest of the AHB system can ignore the ARM9E-S core and function as normal. Because the rest of the system continues operation, the core ignores aborts and interrupts.

The **HRESETn** signal must be held stable during debug. If the system applies Reset to the ARM968E-S (**HRESETn** is driven LOW), the ARM9E-S core changes state without the knowledge of the debugger.

## 8.5 ARM9E-S core clock domains

The ARM968E-S processor has a single clock, **CLK**, that is qualified by two clock enables:

- **SYSCLKEN** controls access to the memory system
- **DBGTCKEN** controls debug operations.

During normal operation, **SYSCLKEN** conditions **CLK** to clock the core. When the processor is in debug state, **DBGTCKEN** conditions **CLK** to clock the core.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 8.6 Determining the core and system state

When the ARM968E-S processor is in debug state, you can examine the core and system state by forcing the load and store multiples into the instruction pipeline.

Before you can examine the core and system state, the debugger must determine if the processor entered debug from manual state or manual state, by examining bit 4 of the EmbeddedICE-RT Debug Status Register. When bit 4 is HIGH, the core entered debug from manual state.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.7 About the EmbeddedICE-RT

The EmbeddedICE-RT logic provides integrated on-chip debug support for the ARM9E-S core within the ARM968E-S processor.

EmbeddedICE-RT is programmed serially using the ARM9E-S TAP controller. Figure 8-7 shows the relationship between the processor, EmbeddedICE-RT, and the TAP, showing only the signals that are pertinent to EmbeddedICE-RT.

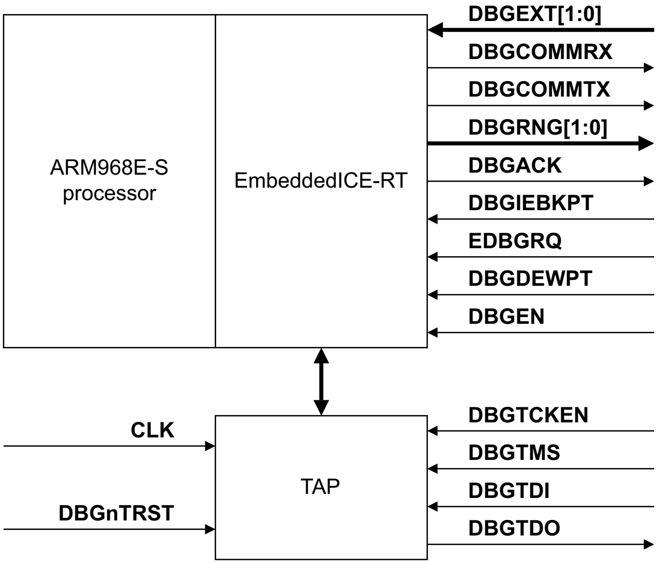


**Figure 8-7 EmbeddedICE-RT interface**

The EmbeddedICE-RT logic contains:

- two real-time watchpoint units
- two independent registers, the Debug Control Register and the Debug Status Register
- debug comms channel.

The Debug Control Register and the Debug Status Register provide overall control of EmbeddedICE-RT operation.

You can program one or both watchpoint units to halt the execution of instructions by the core. Execution halts when the values programmed into EmbeddedICE-RT match the values currently appearing on the address bus, data bus, and various control signals.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Note**

Any bit can be masked so that its value does not affect the comparison.

Each watchpoint unit can be configured to be either a watchpoint to monitor data accesses or a breakpoint to monitor instruction fetches. Watchpoints and breakpoints can be data-dependent.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.8 Disabling EmbeddedICE-RT

You can disable EmbeddedICE-RT by pulling the **DBGEN** input LOW.

**Caution**

Permanently tying the **DBGEN** input LOW disables debug access.

Pulling **DBGEN** LOW disables **DBGDEWPT**, **DBGIEBKPT**, and **EDBGRQ** to the processor and forces **DBGACK** from the processor LOW.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 8.9 The debug comms channel

The EmbeddedICE-RT logic contains a debug comms channel for passing information between the target and the host debugger. The debug comms channel is implemented as *CoProcessor 14* (CP14).

### 8.9.1 Debug comms channel registers

The debug comms channel consists of:

- a 6-bit Debug Comms Channel Status Register
- a 32-bit Debug Comms Channel Data Read Register
- a 32-bit Debug Comms Channel Data Write Register
- a 1-bit Debug Comms Channel Monitor Debug-Mode Status Register.

These registers are located in fixed locations in the EmbeddedICE-RT logic register map. From the viewpoint of the debugger, the registers are accessed using the debug data chain in the usual way. From the viewpoint of the processor, these registers are accessed using MCR and MRC instructions to CP14.

**Note**

Because the Thumb instruction set does not contain coprocessor instructions, use SWI instructions to access debug comms channel data when in Thumb state.

Table 8-3 lists the registers of the debug comms channel.

**Table 8-3 Debug comms channel registers**

| Register name | Access |
|---|---|
| CP14 c0 Debug Comms Channel Status Register | Read-only |
| CP14 c1 Debug Comms Channel Data Read Register | Read-only |
| CP14 c1 Debug Comms Channel Data Write Register | Write-only |
| CP14 c2 Debug Comms Channel Monitor Debug-Mode Status Register | Read/write |

#### Debug Comms Channel Status Register

The read-only Debug Comms Channel Status Register is a handshake register between the processor and the asynchronous debugger. Use the following instruction to read the Comms Channel Status Register:

```
MRC p14, 0, <Rd>, c0, 0 ; Read Debug Comms Channel Status Register
```

Figure 8-8 shows the bit fields of the Debug Comms Channel Status Register.

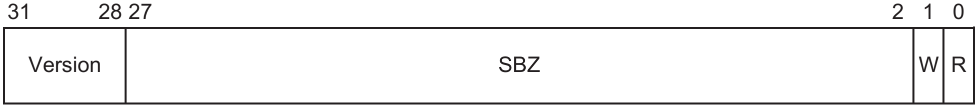


**Figure 8-8 Debug Comms Channel Status Register**

Table 8-4 lists the bit field definitions of the Debug Comms Channel Status Register.

**Table 8-4 Debug Comms Channel Status Register Encoding**

| Bit | Name | Definition |
|---|---|---|
| [31:28] | Version | EmbeddedICE-RT version number, b0110. |
| [27:2] | - | Reserved. Should Be Zero. |
| [1] | W | Comms Channel Data Write Register ready flag:<br>1 = data ready for scan-out<br>0 = channel ready for new data from processor. |
| [0] | R | Comms Channel Data Read Register ready flag:<br>1 = data ready for processor to read<br>0 = channel ready for new data from debugger. |

### Debug Comms Channel Read Data Register

Use the following instruction to read data from the 32-bit read-only Debug Comms Channel Data Read Register:

```
MRC p14, 0, <Rd>, c1, c0
```

### Debug Comms Channel Write Data Register

Use the following instruction to write data to the 32-bit write-only Debug Comms Channel Data Write Register:

```
MCR p14, 0, <Rn>, c1, c0
```

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Debug Comms Channel Monitor Debug-Mode Status Register

A debug monitor can use the read/write Debug Comms Channel Monitor Debug-Mode Status Register when the ARM968E-S processor is in monitor debug-mode. Use the instructions in Table 8-5 to access the Debug Comms Channel Monitor Debug-Mode Register:

**Table 8-5 Debug Comms Channel Monitor Debug-Mode Status Register instructions**

| Instruction | Operation |
|---|---|
| `MRC p14, 0, <Rd>, c2, c0` | Read Debug Comms Channel Monitor Mode-Debug Status Register |
| `MCR p14, 0, <Rd>, c2, c0` | Write Debug Comms Channel Monitor Mode-Debug Status Register |

Figure 8-9 shows the bit fields of the Debug Comms Channel Monitor Debug-Mode Status Register.

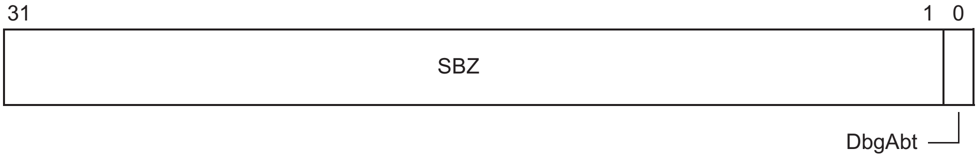


**Figure 8-9 Debug Comms Channel Monitor Debug-Mode Status Register**

Table 8-6 lists the bit field definitions of the Debug Comms Channel Monitor Debug-Mode Status Register.

**Table 8-6 Debug Comms Channel Monitor Debug-Mode Status Register Encoding**

| Bit | Name | Definition |
|---|---|---|
| [0] | DbgAbt | Debug mode abort bit:<br>1 = breakpoint or watchpoint caused Prefetch Abort or Data Abort<br>0 = no abort on breakpoint or watchpoint. |
| [31:1] | - | Should Be Zero. |

If both the debug abort and external abort signals are asserted on an instruction fetch or data access, the external abort takes priority, and the DbgAbt bit is not set.

A real-time debug-aware abort handler can examine bit 0 to determine if the abort is externally or internally generated. If the DbgAbt bit is set, the abort handler initiates communication with the debugger over the debug comms channel.

### 8.9.2 Communications using the debug comms channel

Messages can be sent and received using the debug comms channel as described in:

- *Sending a message to the debugger*
- *Receiving a message from the debugger*.

#### Sending a message to the debugger

Before sending a message to the debugger, the processor must first check the W bit in the Debug Comms Channel Status Register.

If the W bit is clear, the Debug Comms Channel Write Data Register is ready for new write data. The processor can write to the Debug Comms Channel Write Data Register with a register transfer to CP14. The register transfer sets the W bit.

If the W bit is set, the debugger has not read the previously written data. The processor must continue polling the Debug Comms Channel Status Register until the W bit is clear.

The debugger polls the Debug Comms Channel Status Register through the JTAG interface. When it detects that the W bit is set, it can read the Debug Comms Channel Write Data Register and scan the data out. This action clears the W bit, and the processor can send another message to the debugger.

#### Receiving a message from the debugger

Transferring a message from the debugger to the processor is similar to sending a message to the debugger. In this case, the debugger must first check the R bit in the Debug Comms Channel Status Register.

If the R bit is clear, the Debug Comms Channel Read Data Register is ready, and the debugger can write to the Debug Comms Channel Read Data Register using the JTAG interface. The write sets the R bit.

If the R bit is set, the processor has not read the previously written data. The debugger must continue polling the Debug Comms Channel Read Data Register until the R bit is clear.

The processor polls the Debug Comms Channel Status Register. When it detects that the R bit is set, it can read the Debug Comms Channel Read Data Register with an MRC instruction to CP14. This action clears the R bit, and the debugger can send another message to the processor.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.10 Monitor debug-mode

The ARM9E-S core contains logic that enables debugging of a system without stopping the core entirely. Critical interrupt routines can continue while the core is being interrogated by the debugger. Setting bit 4 of the Debug Control Register enables the real-time debug features of the ARM9E-S core. Setting this bit configures the EmbeddedICE-RT logic so that a breakpoint or watchpoint causes the processor to enter Abort mode, taking the Prefetch Abort or Data Abort vectors respectively. When the processor is configured for real-time debugging you must be aware of the following restrictions:

- Breakpoints or watchpoints cannot be data-dependent. Chaining is supported, but no support is provided for use of the range functionality. Breakpoints or watchpoints can only be based on:
  - instruction or data addresses
  - external watchpoint conditioner (**DBGEXTERN**)
  - user or privileged mode access (**DnTRANS** and **InTRANS**)
  - read or write access (watchpoints)
  - access size (breakpoints, **ITBIT**, and watchpoints, **DMAS[1:0]**).
- The single-step hardware is not enabled.
- External breakpoints and watchpoints are not supported.
- The vector catching hardware can be used but must not be configured to catch the Prefetch or Data Abort exceptions.

**Caution**

No support is provided to mix halt mode and monitor debug-mode functionality. When the core is configured into the monitor debug-mode, asserting the external **EDBGRQ** signal causes Unpredictable behavior. Setting the internal **EDBGRQ** bit results in Unpredictable behavior.

When an abort is generated by the monitor debug-mode, it is recorded in the Debug Status Register in CP14. See *Debug Comms Channel Monitor Debug-Mode Status Register* on page 8-20.

Because the monitor debug-mode does not put the ARM9E-S core into debug state, it is necessary to change the contents of the watchpoint registers while TCM accesses are taking place, rather than being changed when in debug state. Writing to a watchpoint register during an access disables all matches from the watchpoint unit using the register for the cycle of the update.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

If there is a possibility of false matches occurring during changes to the watchpoint registers, then you must:

1. Disable that watchpoint unit using the control register for that watchpoint unit.
2. Change the other registers.
3. Re-enable the watchpoint unit by rewriting the control register.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 8.11 Additional debug reading

See the *ARM9E-S Technical Reference Manual*, Appendix C Debug in Depth for a full description of the ARM9E-S debug features and the JTAG interface.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 9
# Embedded Trace Macrocell Interface

This chapter describes the ARM968E-S *Embedded Trace Macrocell* (ETM) interface. It contains the following sections:

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 9.1 About the ETM interface

The full ARM968E-S debug implementation supports the connection of an external *Embedded Trace Module* (ETM) to provide real time code tracing of the processor in an embedded system. See the *ETM9 Technical Reference Manual* for more information.

The ETM interface is primarily one way. To provide code tracing, the ETM block must be able to monitor various ARM9E-S inputs and outputs. The ETM interface drives the required ARM9E-S inputs and outputs out from the processor through the ETM interface registers, as Figure 9-1 shows.

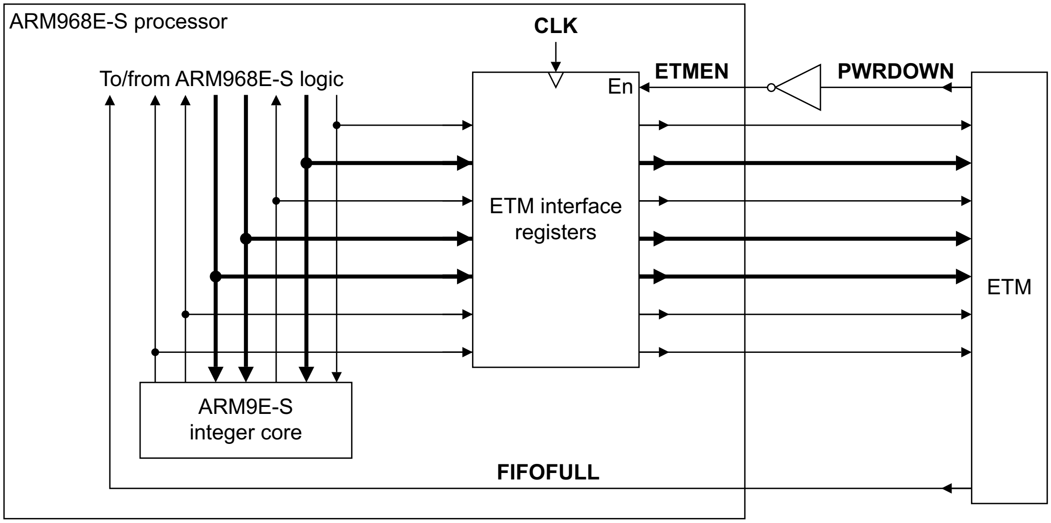


**Figure 9-1 ETM interface**

The pipelined outputs of the ETM interface provide early output timing and isolate any ETM input load from the critical processor signals. Because all outputs have the same delay, the latency of the pipelined outputs does not affect ETM trace behavior.

## 9.2 Enabling the ETM interface

The top-level pin **ETMEN** enables the ETM interface. When this input is HIGH, the ETM interface drives the outputs so that an external ETM can begin code tracing.

When the **ETMEN** input is LOW, the ETM interface outputs remain at their last value before the interface was disabled.

The ETM usually drives the **ETMEN** input HIGH after the debugger programs the ETM using its TAP controller.

ARM recommends that you connect the **ETMEN** input to the **PWRDOWN** output of the ETM9 through an inverter as Figure 9-1 on page 9-2 shows.

**Note**

If your design does not include an ETM, tie the **ETMEN** input LOW to save power.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 9.3 Trace support features

The trace support uses the following features:

- *FIFOFULL*
- *Configuration Control Register*
- *Trace Process ID Register*.

### 9.3.1 FIFOFULL

The ETM9 drives the **FIFOFULL** input to the processor when the ETM FIFO contents reach the programmed upper watermark. The processor uses **FIFOFULL** to stall the ARM9E-S core, preventing trace loss. The core remains stalled until **FIFOFULL** is deasserted.

The processor can stall only on instruction boundaries, enabling any current AHB transfers to complete. You must take this into consideration when programming the ETM FIFO watermark. If the current instruction is either LDM or STM, the FIFO might have to accept up to 16 words after the assertion of **FIFOFULL**.

——— **Note** ———

Using **FIFOFULL** to stall the processor affects real-time operating performance.

———————————

### 9.3.2 Configuration Control Register

The Configuration Control Register enables masking of the **nIRQ** and **nFIQ** interrupt signals when the ETM FIFO is full. See *CP15 c15 Configuration Control Register* on page 4-11 for a description of this register.

### 9.3.3 Trace Process ID Register

The Trace Process ID Register enables real-time trace tools to identify the currently executing process in multitasking environments. See *CP15 c13 Trace Process ID Register* on page 4-11 for a description of this register.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 10
# Test Support

This chapter describes the test methodology employed for the ARM968E-S synthesized logic and tightly-coupled memory. It contains the following sections:

- *About the ARM968E-S test methodology* on page 10-2
- *Scan insertion and ATPG* on page 10-3.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## 10.1 About the ARM968E-S test methodology

To achieve a high level of fault coverage, the ARM9E-S core and ARM968E-S control logic implement scan insertion and ATPG techniques as part of the synthesis flow.

## 10.2 Scan insertion and ATPG

This technique is covered in detail in the *ARM968E-S Implementation Guide*. Scan insertion requires all register elements to be replaced by scannable versions that are then connected into a number of large scan chains. These scan chains are used to set up data patterns on the combinatorial logic between the registers and to capture the logic outputs. The logic outputs are then scanned out while the next data pattern is scanned in.

*Automated Test Pattern Generation* (ATPG) tools are used to create the necessary scan patterns to test the logic when the scan insertion has been performed. This technique enables very high fault coverage of the standard cell combinatorial logic, typically in the 95-99% range.

Scan insertion does have an impact on the area and performance of the synthesized design, because of the larger scan register elements and the serial routing between them. To minimize these effects the scan insertion is performed early in the synthesis cycle and the design re-optimized with the scan elements in place.

### 10.2.1 ARM968E-S test wrapper

A test wrapper can be used to improve test coverage where an ASIC contains a black-box macrocell with no internal visibility of the macrocell. For logic to be testable, the input to the logic must be controllable using a scan chain and so must be driven by a register. The output of the logic must also be observable through a scan chain, and so must be registered.

If the processor is integrated into an ASIC as a black box, the test tools do not have visibility of the internal scan chains and cannot create vectors to cover any logic between the last register in the ARM968E-S processor and the next register in the ASIC. This is known as a test shadow and leads to a reduction in test coverage.

The test wrapper enables testing of this shadow logic. The test wrapper is a scan chain around the periphery of the processor that connects to each input and output. The test wrapper scan chain can be used in two modes:

- INTEST mode
- EXTEST mode.

The INTEST wrapper is required only for embedded ARM968E-S processors. In INTEST mode, all processor inputs are driven using the test wrapper scan chain, and all processor outputs are observable through the test wrapper scan chain. This enables a complete set of ATPG vectors to be produced for the processor in isolation. In EXTEST mode, all processor inputs are driven using the test wrapper scan chain. All processor inputs are observable through the test wrapper scan chain. This enables the logic surrounding the processor to be tested without requiring internal visibility of the processor.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Chapter 11
# DFT Interface

This chapter describes the *Design for Test* (DFT) interface of the ARM968E-S processor. It contains the following section:

- *About the DFT interface* on page 11-2.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## 11.1 About the DFT interface

The Synopsys Reference Methodology flow contains optional scripts to create test wrappers. The *Synopsys Methodology Guide* in the `<technology>/synopsys/docs` directory contains details of the structure and operation of the test wrappers.

The ARM968E-S DFT interface has two dedicated DFT signals:

**SE** Scan enable. **SE** disables the TCM chip select to ensure TCM state preservation during *Automated Test Pattern Generation* (ATPG). Connect **SE** to the shift enable of your design.

**WEXTEST** Wrapper external test select. **WEXTEST** prevents unknown states during EXTEST by forcing clock gates of shared wrapper cells to always be enabled. **WEXTEST** must be HIGH during EXTEST or LOW during INTEST or functional mode.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Appendix A
# Signal Descriptions

This appendix describes the ARM968E-S signals. It contains the following sections:

- *Signal properties and requirements* on page A-2
- *AHB interface signals* on page A-3
- *DMA interface signals* on page A-5
- *Debug signals* on page A-7
- *TCM interface signals* on page A-9
- *ETM interface signals* on page A-12
- *DFT interface signals* on page A-14
- *Miscellaneous interface signals* on page A-15.

## A.1 Signal properties and requirements

For easy integration of the ARM968E-S processor into embedded applications and to simplify synthesis flow, the ARM968E-S design uses the following techniques:

- a single rising edge clock times all activity
- unidirectional signals and buses
- inputs that are required to be synchronous to the single clock.

All outputs change from the rising clock edge, and all input sampling occurs on the rising clock edge. These techniques simplify the definition of the top-level ARM968E-S signals. In addition, all signals are either input-only or output-only. There are no bidirectional signals.

**Note**

Asynchronous signals, such as interrupt sources, must first be synchronized by external logic before being applied to the processor.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## A.2 AHB interface signals

Table A-1 describes the ARM968E-S AHB interface signals.

**Table A-1 AHB interface signals**

| Name | Direction | Description |
|---|---|---|
| **HADDR[31:0]** | Output | Address bus. |
| **HTRANS[1:0]** | Output | Transfer type:<br>b00 = idle<br>b10 = nonsequential<br>b11 = sequential. |
| **HBURST[2:0]** | Output | BIU burst type:<br>b000 = single transfer<br>b001 = incrementing transfer of unspecified length<br>b011 = 4-beat incrementing burst<br>b101 = 8-beat incrementing burst<br>b111 = 16-beat incrementing burst. |
| **HWRITE** | Output | Transfer direction:<br>1 = write<br>0 = read. |
| **HSIZE[2:0]** | Output | Transfer size:<br>b000 = byte<br>b001 = halfword<br>b010 = word. |
| **HPROT[3:0]** | Output | BIU protection control:<br>bxxx0 = opcode fetch<br>bxxx1 = data access<br>bxx0x = User mode access<br>bxx1x = Supervisor mode access<br>bx0xx = nonbufferable access<br>bx1xx = bufferable access<br>b0xxx = noncachable access<br>b1xxx = cachable access. |
| **HREADY** | Input | Slave ready. Can be driven LOW to extend transfer. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Table A-1 AHB interface signals (continued)

| Name | Direction | Description |
|---|---|---|
| **HRESP** | Input | Slave response. Reflects transfer status:<br>0 = okay<br>1 = error. |
| **HWDATA[31:0]** | Output | Write data bus. |
| **HRDATA[31:0]** | Input | Read data bus. |
| **HMASTLOCK** | Output | Bus locked. Indicates that processor has locked access to AHB bus. Asserted when executing SWP instructions to AHB address space. |
| **HRESETn** | Input | Active-LOW system and bus reset. Asserted asynchronously. Deasserted synchronously. |
| **HCLKEN** | Input | Synchronous **HCLK** enable. Specifies rising edge of **HCLK** for AHB transfer. If **CLK** and **HCLK** have same frequency, tie **HCLKEN** HIGH. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## A.3 DMA interface signals

Table A-2 describes the ARM968E-S DMA interface signals.

**Table A-2 DMA interface signals**

| Name | Direction | Description |
|---|---|---|
| **HSELD** | Input | DMA select:<br>1 = DMA selected<br>0 = DMA not selected. |
| **HADDRD[31:0]** | Input | DMA address bus. |
| **HTRANSD[1:0]** | Input | DMA transfer type:<br>b00 = idle<br>b01 = busy<br>b10 = nonsequential<br>b11 = sequential. |
| **HBURSTD[2:0]** | Input | These burst type signals are not implemented and are present only for AMBA specification compliance. |
| **HWRITED** | Input | DMA transfer direction:<br>1 = write to DMA<br>0 = read from DMA. |
| **HSIZED[2:0]** | Input | DMA transfer size:<br>b000 = byte<br>b001 = halfword<br>b010 = word. |
| **HPROTD[3:0]** | Input | These protection control signals are not implemented and are present only for AMBA specification compliance. |
| **HREADYIND** | Input | Slave ready input:<br>1 = slave transfer complete<br>0 = slave transfer in progress |
| **HWDATAD[31:0]** | Input | DMA write data bus. |
| **HREADYOUTD** | Output | DMA ready output:<br>1 = DMA transfer complete<br>0 = DMA transfer in progress. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Table A-2 DMA interface signals (continued)

| Name | Direction | Description |
|---|---|---|
| **HRESPD** | Output | DMA response. Reflects transfer status:<br>1 = error<br>0 = okay. |
| **HRDATAD[31:0]** | Output | DMA read data bus. |
| **HCLKEND** | Input | Clock enable for DMA interface. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## A.4 Debug signals

Table A-3 describes the ARM968E-S debug signals.

——— **Note** ———

**DBGRNG**, **DBGIEBKPT**, and **DBGDEWPT** are present only in the full debug configuration.

**Table A-3 Debug signals**

| Name | Direction | Description |
|---|---|---|
| **COMMRX** | Output | Comms channel receive. HIGH when comms channel receive buffer has data to read. |
| **COMMTX** | Output | Comms channel transmit. HIGH when comms channel transmit buffer is empty. |
| **DBGACK** | Output | Debug acknowledge. HIGH when processor is in debug state. |
| **DBGEN** | Input | Debug enable. HIGH when processor is in debug state. Enables EmbeddedICE logic. |
| **DBGRQI** | Output | Internal debug request. Represents the debug request signal that is presented to the core debug logic. This is a combination of EDBGRQ and bit 1 of the Debug Control Register. |
| **EDBGRQ** | Input | External debug request from external debugger. |
| **DBGEXT[1:0]** | Input | EmbeddedICE external input. Conditions breakpoints and watchpoints. |
| **DBGINSTREXEC** | Output | Instruction executed. HIGH when instruction in Execute stage is done. |
| **DBGRNG[1:0]** | Output | Debug rangeout. Indicates that the corresponding EmbeddedICE-RT watchpoint register matches the conditions currently present on the address, data and control buses. This signal is independent of the state of the watchpoint enable control bit. |
| **DBGIEBKPT** | Input | Instruction breakpoint. Asserted by external hardware to halt execution of the processor for debug. If HIGH at the end of an instruction fetch, it causes the processor to enter debug state when the instruction reaches the Execute stage. |
| **DBGDEWPT** | Input | Data watchpoint. Asserted by external hardware to halt execution of the processor for debug. If HIGH at the end of a data memory request cycle, it causes the processor to enter debug state. |
| **DBGnTRST** | Input | Test reset. Active-LOW reset signal for the EmbeddedICE internal state. Can be asserted asynchronously but must be deasserted synchronously. |
| **DBGTCKEN** | Input | Synchronous test clock enable for debug logic accessed by JTAG interface. When HIGH on rising edge of **CLK**, debug logic is able to advance. |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Table A-3 Debug signals (continued)**

| Name | Direction | Description |
|---|---|---|
| **DBGTDI** | Input | Test data input for debug logic. |
| **DBGTMS** | Input | Test mode select for TAP controller. |
| **DBGTDO** | Output | Test data output from debug logic. |
| **DBGIR[3:0]** | Output | TAP Controller Instruction Register. Reflects current instruction loaded in the TAP Controller Control Register. Changes when TAP controller is in UPDATE-IR state. |
| **DBGSCREG[4:0]** | Output | Scan chain register. Reflects ID number of scan chain currently selected by TAP controller. Changes when TAP controller is in UPDATE-DR state. |
| **DBGTAPSM[3:0]** | Output | TAP controller state machine. Reflects current state of TAP controller state machine. |
| **DBGnTDOEN** | Output | Debug output enable. Active-LOW enable indicates that serial data is being driven out of **DBGTDO** output. |
| **DBGSDIN** | Output | Serial data in. Contains serial data to be applied to an external scan chain. |
| **DBGSDOUT** | Input | Serial data out. Contains serial data out of an external scan chain. Must be LOW when no external scan chain is connected. |
| **TAPID[31:0]** | Input | Boundary scan ID code. Shifts out ID code on **DBGTDO** when IDCODE instruction is entered into TAP controller. |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## A.5 TCM interface signals

This section contains the following ARM968E-S signal descriptions:

- *DTCM0 and DTCM1 interface signals*
- *ITCM interface signals* on page A-10.

### A.5.1 DTCM0 and DTCM1 interface signals

Table A-4 shows the data ARM968E-S TCM interface signals.

**Table A-4 DTCM0 and DTCM1 interface signals**

| Signal | Direction | Function |
|---|---|---|
| **D0TCMADDR[21:3]** | Output | DTCM0 address bus. Addresses up to 4MB. |
| **D0TCMWD[31:0]** | Output | DTCM0 write data bus. |
| **D0TCMCS** | Output | DTCM0 chip select. |
| **D0TCMWE[3:0]** | Output | DTCM0 byte write enable. Each set bit indicates a write to RAM of the corresponding byte in **D0TCMWD[31:0]**. For example:<br>b0000 = No write.<br>b0001 = Byte write of the least significant byte.<br>b1000 = Byte write of the most significant byte.<br>b0011 = A half-word write of the least significant two bytes.<br>**D0TCMWE[3:0]** bits are set only when a write is taking place, so when **D0TCMnRW** is not set, **D0TCMWE[3:0]** = b0000. |
| **D0TCMnRW** | Output | DTCM0 read/write:<br>1 = write access<br>0 = read access. |
| **D0TCMRD[31:0**] | Input | DTCM0 read data. |
| **D0TCMWAIT** | Input | DTCM0 wait. If HIGH, ITCM cannot service any requests in next cycle. Stall the processor for multiple-cycle-access RAM on DTCM0 interface. Used to stall the processor for DMA access to single-port DTCM0. |
| **D0TCMERROR** | Input | DTCM0 error signal. Enables the processor to read error conditions during read accesses. |
| **D1TCMADDR[21:3]** | Output | DTCM1 address. Addresses up to 4MB. |
| **D1TCMWD[31:0]** | Output | DTCM1 write data. |
| **D1TCMCS** | Output | DTCM1 chip select. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Table A-4 DTCM0 and DTCM1 interface signals (continued)**

| Signal | Direction | Function |
|---|---|---|
| **D1TCMWE[3:0]** | Output | DTCM1 byte write enable. Each bit indicates a write to RAM of the corresponding byte in **D1TCMWD[31:0]**. For example:<br>b0000 = No write.<br>b0001 = Byte write of the least significant byte.<br>b1000 = Byte write of the most significant byte.<br>b0011 = A half-word write of the least significant two bytes.<br>**D1TCMWE[3:0]** bits are set only when a write is taking place, so when **D1TCMnRW** is not set, **D1TCMWE[3:0]** = b0000. |
| **D1TCMnRW** | Output | DTCM1 read/write:<br>1 = write access<br>0 = read access. |
| **D1TCMRD[31:0**] | Input | DTCM1 read data. |
| **D1TCMWAIT** | Input | DTCM1 wait. If HIGH, ITCM cannot service any requests in next cycle. Stall the processor for multiple-cycle-access RAM on DTCM1 interface. Used to stall the processor for DMA access to single-port DTCM1. |
| **D1TCMERROR** | Input | DTCM1 error signal. Enables the processor to read error conditions during read accesses. |
| **DTCMSIZE[4:0]** | Input | DTCM0 and DTCM1 size:<br>b00000 = 0B<br>b00001 = 1KB<br>b00010 = 2KB<br>b00011 = 4KB<br>b00100 = 8KB<br>b00101 = 16KB<br>b00110 = 32KB<br>b00111 = 64KB<br>b01000 = 128KB<br>b01001 = 256KB<br>b01010 = 512KB<br>b01011 = 1MB<br>b01100 = 2MB<br>b01101 = 4MB.<br>The supported sizes are 0 and $2^n$KB for n = 0-12. |

## A.5.2 ITCM interface signals

Table A-5 on page A-11 shows the ARM968E-S ITCM interface signals.

**Table A-5 ITCM interface signals**

| Signal | Direction | Function |
|---|---|---|
| **ITCMADDR[21:2]** | Output | ITCM address. Addresses up to 4MB. Output delay 90% of clock cycle. |
| **ITCMWD[31:0]** | Output | ITCM write data. |
| **ITCMCS** | Output | ITCM chip select. |
| **ITCMWE[3:0]** | Output | ITCM byte write enable. Each bit indicates a write to RAM of the corresponding byte in **ITCMWD[31:0]**. For example:<br>b0000 = No write.<br>b0001 = Byte write of the least significant byte.<br>b1000 = Byte write of the most significant byte.<br>b0011 = A half-word write of the least significant two bytes.<br>**ITCMWE[3:0]** bits are set only when a write is taking place, so when **ITCMnRW** is not set, **ITCMWE[3:0]** = b0000. |
| **ITCMnRW** | Output | ITCM read/write:<br>1 = write access<br>0 = read access. |
| **ITCMRD[31:0**] | Input | Instruction TCM read data. |
| **ITCMWAIT** | Input | ITCM wait. If HIGH, ITCM cannot service any requests in next cycle. Stall the processor for multiple-cycle-access RAM on ITCM interface. Used to stall the processor for DMA access to single-port ITCM. |
| **ITCMSIZE[4:0]** | Input | ITCM size:<br>b00000 = 0B<br>b00001 = 1KB<br>b00010 = 2KB<br>b00011 = 4KB<br>b00100 = 8KB<br>b00101 = 16KB<br>b00110 = 32KB<br>b00111 = 64KB<br>b01000 = 128KB<br>b01001 = 256KB<br>b01010 = 512KB<br>b01011 = 1MB<br>b01100 = 2MB<br>b01101 = 4MB.<br>The supported sizes are 0 and $2^n$KB for n = 0-12. |
| **ITCMERROR** | Input | ITCM error signal. Enables the processor to read error conditions during read accesses. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## A.6 ETM interface signals

Table A-6 describes the ARM968E-S ETM interface signals.

——— **Note** ———

ETM interface signals are present only in the full debug configuration.

**Table A-6 ETM interface signals**

| Name | Direction | Description |
|---|---|---|
| **ETMBIGEND** | Output | Big-endian configuration indication. |
| **ETMHIVECS** | Output | Exception vectors configuration. |
| **ETMnWAIT** | Output | Processor stalled indication. |
| **ETMIA[31:1]** | Output | Instruction address. |
| **ETMInMREQ** | Output | Instruction memory request. |
| **ETMISEQ** | Output | Sequential instruction access. |
| **ETMITBIT** | Output | Thumb state indication. |
| **ETMID31To25[31:25]** | Output | Instruction data field. |
| **ETMID15To11[15:11]** | Output | Instruction data field. |
| **ETMDA[31:0]** | Output | Data address. |
| **ETMWDATA[31:0]** | Output | Write data. |
| **ETMDMAS[1:0]** | Output | Data size indication. |
| **ETMDnMREQ** | Output | Data memory request. |
| **ETMDnRW** | Output | Data not read or write. |
| **ETMDSEQ** | Output | Sequential data indication. |
| **ETMRDATA[31:0]** | Output | Read data. |
| **ETMDABORT** | Output | Data Abort. |
| **ETMCHSD[1:0]** | Output | Coprocessor handshake decode signals. |
| **ETMCHSE[1:0]** | Output | Coprocessor handshake execute signals. |
| **ETMLATECANCEL** | Output | Coprocessor late cancel indication. |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

Table A-6 ETM interface signals (continued)

| Name | Direction | Description |
|---|---|---|
| **ETMPASS** | Output | Coprocessor instruction execute indication. |
| **ETMDBGACK** | Output | Debug state indication. |
| **ETMINSTREXEC** | Output | Instruction execute indication. |
| **ETMINSTRVALID** | Output | Instruction valid indication. |
| **ETMRNGOUT[1:0]** | Output | Watchpoint register match indication. |
| **ETMPROCID[31:0]** | Output | Process ID. |
| **ETMPROCIDWR** | Output | Asserted when **ETMPROCID** is written. |
| **ETMEN** | Input | Synchronous ETM interface enable. Must be LOW if ETM is not used. |
| **FIFOFULL** | Input | Asserted when ETM FIFO fills. Must be LOW if ETM is not used. |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## A.7 DFT interface signals

Table A-7 describes the ARM968E-S DFT signals.

**Table A-7 DFT signals**

| Name | Direction | Description |
|---|---|---|
| **SE** | Input | Scan enable. HIGH = shift. Must be tied LOW during functional operation. |
| **WEXTEST** | Input | Wrapper EXTEST select. If the optional test wrapper exists, **WEXTEST** selects the test path from the wrapper:<br>1 = EXTEST mode<br>0 = INTEST mode or functional mode.<br>**WEXTEST** also ensures that the wrapper clock gates are active during test mode. Must be tied LOW during functional operation. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## A.8 Miscellaneous interface signals

Table A-8 describes the ARM968E-S interface signals not included in the other tables in this chapter.

**Table A-8 Miscellaneous interface signals**

| Name | Direction | Description |
|---|---|---|
| **CLK** | Input | System clock. Times all processor operations. All outputs change on rising edge. All inputs sampled on rising edge. **CLK** can be stretched in either phase. |
| **nFIQ** | Input | LOW-active fast interrupt request:<br>0 = fast interrupt request<br>1 = no fast interrupt request.<br>**nFIQ** must be synchronous with **CLK**. |
| **nIRQ** | Input | LOW-active Interrupt request:<br>0 = interrupt request<br>1 = no interrupt request.<br>**nIRQ** must be synchronous with **CLK**. |
| **VINITHI** | Input | High exception vector address select:<br>1 = vector addresses start at `0xFFFF0000`<br>0 = vector addresses start at `0x00000000`.<br>The state of **VINITHI** at Reset determines vector locations. |
| **INITRAM** | Input | TCM enable:<br>1 = TCMs enabled<br>0 = TCMs disabled.<br>The state of **INITRAM** at Reset enables or disables TCMs. |
| **BIGENDOUT** | Output | Big-endian select:<br>1 = memory mapped as big-endian<br>0 = memory mapped as little-endian. |
| **STANDBYWFI** | Output | Wait-for-interrupt flag:<br>1 = processor in wait-for-interrupt mode<br>0 = processor in normal operating mode. |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

# Appendix B
# **AC Parameters**

This appendix describes the AC timing parameters for the ARM968E-S processor. It contains the following sections:

## B.1 About AC timing parameters

All figures are expressed as percentages of the **CLK** period at maximum operating frequency.

The figures quoted are relative to the rising clock edge after the clock skew for internal buffering has been added. Inputs given a 0% hold figure therefore require a positive hold relative to the top-level clock input. The amount of hold required is equivalent to the internal clock skew.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.2 CLK, HCLKEN, and HRESETn timing parameters

Figure B-1 shows the setup time and hold time parameters of the **HCLKEN** and **HRESETn** signals.

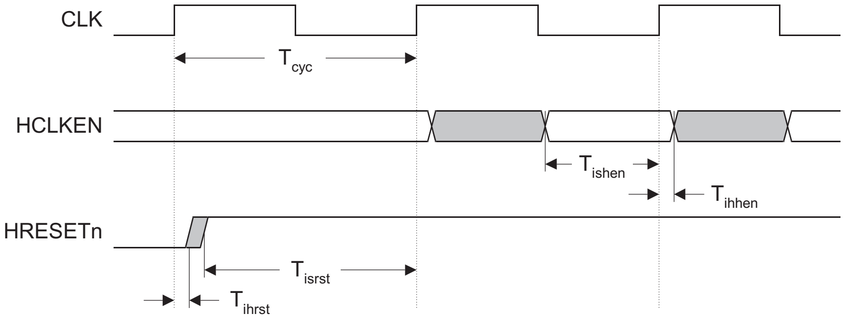


**Figure B-1 CLK, HCLKEN, and HRESETn timing parameters**

Table B-1 describes the timing parameters shown in Figure B-1.

**Table B-1 CLK, HCLKEN, and HRESETn timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{cyc}$ | **CLK** cycle time | 100% | - |
| $T_{ishen}$ | **HCLKEN** input setup time to rising **CLK** | 85% | - |
| $T_{ihhen}$ | **HCLKEN** input hold time from rising **CLK** | - | 0% |
| $T_{isrst}$ | **HRESETn** deassertion input setup time to rising **CLK** | 90% | - |
| $T_{ihrst}$ | **HRESETn** deassertion input hold time from rising **CLK** | - | 0% |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.3 AHB bus master timing parameters

Figure B-2 shows the AHB bus master timing parameters.

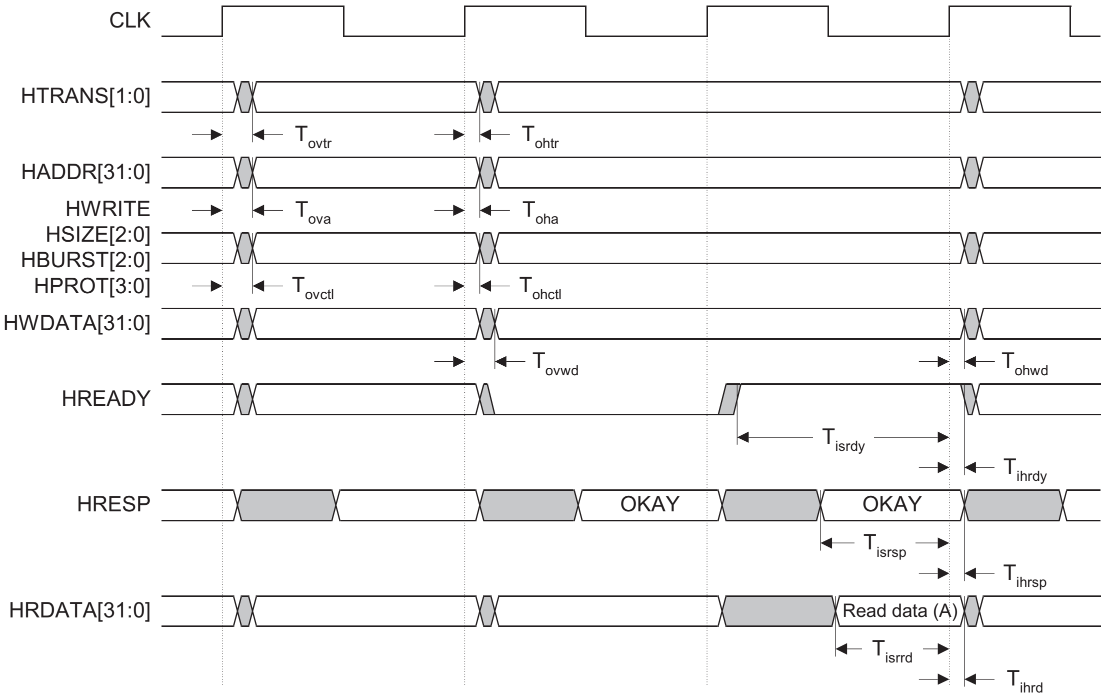


**Figure B-2 AHB bus master timing parameters**

Table B-2 describes the timing parameters shown in Figure B-2.

**Table B-2 AHB bus master timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovtr}$ | Rising **CLK** to **HTRANS[1:0]** valid | - | 30% |
| $T_{ohtr}$ | **HTRANS[1:0]** hold time from rising **CLK** | >0% | - |
| $T_{ova}$ | Rising **CLK** to **HADDR[31:0]** valid | - | 30% |
| $T_{oha}$ | **HADDR[31:0]** hold time from rising **CLK** | >0% | - |
| $T_{ovctl}$ | Rising **CLK** to AHB control signals valid | - | 30% |
| $T_{ohctl}$ | AHB control signals hold time from rising **CLK** | >0% | - |

**Table B-2 AHB bus master timing parameters (continued)**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovwd}$ | Rising **CLK** to **HWDATA[31:0]** valid | - | 30% |
| $T_{ohwd}$ | **HWDATA[31:0]** hold time from rising **CLK** | >0% | - |
| $T_{isrdy}$ | **HREADY** input setup time to rising **CLK** | 40% | - |
| $T_{ihrdy}$ | **HREADY** input hold time from rising **CLK** | - | 0% |
| $T_{isrsp}$ | **HRESP** input setup time to rising **CLK** | 40% | - |
| $T_{ihrsp}$ | **HRESP** input hold time from rising **CLK** | - | 0% |
| $T_{isrd}$ | **HRDATA[31:0]** input setup time to rising **CLK** | 30% | - |
| $T_{ihrd}$ | **HRDATA[31:0]** input hold time from rising **CLK** | - | 0% |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## B.4 DMA interface timing parameters

Figure B-3 shows the DMA interface timing parameters.

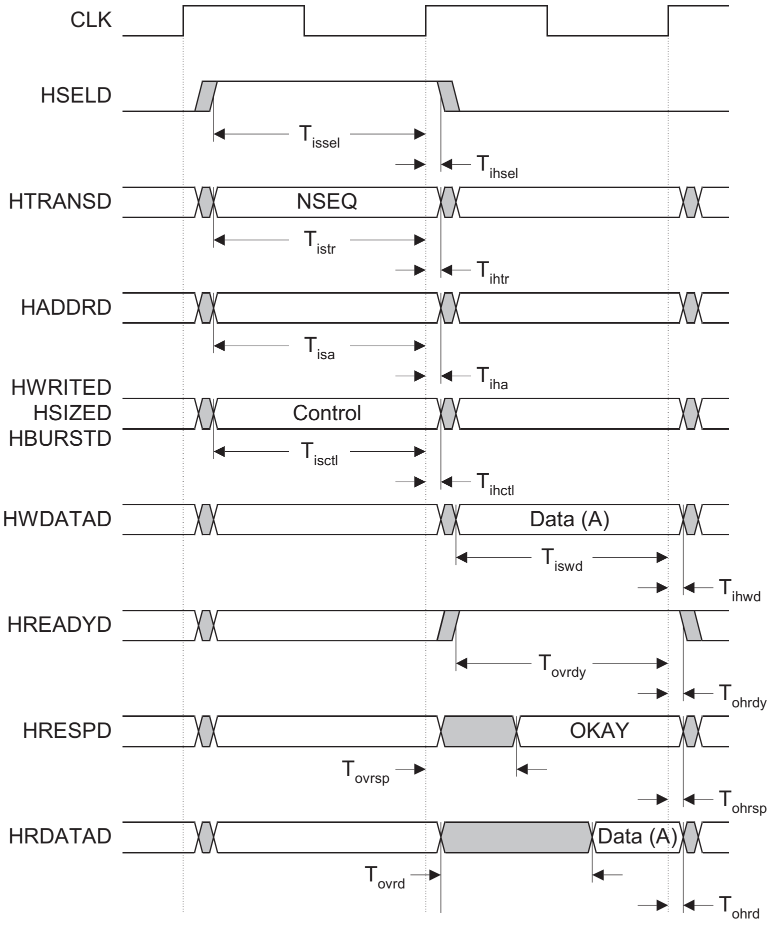


**Figure B-3 DMA interface timing parameters**

Table B-3 on page B-7 describes the timing parameters shown in Figure B-3.

### Table B-3 DMA interface timing parameters

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{issel}$ | **HSELD** setup time before **CLK** | 50% | - |
| $T_{ihsel}$ | **HSELD** hold time after **CLK** | - | 0% |
| $T_{istr}$ | **HTRANSD[1:0]** setup time before **CLK** | 50% | - |
| $T_{ihtr}$ | **HTRANSD[1:0]** hold time after **CLK** | - | 0% |
| $T_{isa}$ | **HADDRD[31:0]** setup time before **CLK** | 50% | - |
| $T_{iha}$ | **HADDRD[31:0]** hold time after **CLK** | - | 0% |
| $T_{isctl}$ | **HWRITED**, **HSIZE[2:0]**, and **HBURSTD[2:0]** setup time before **CLK** | 50% | - |
| $T_{ihctl}$ | **HWRITED**, **HSIZE[2:0]**, and **HBURSTD[2:0]** hold time after **CLK** | - | 0% |
| $T_{iswd}$ | **HWDATAD[31:0]** setup time before **CLK** | 50% | - |
| $T_{ihwd}$ | **HWDATAD[31:0]** hold time after **CLK** | - | 0% |
| $T_{isrdy}$ | **HREADYIND** setup time before **CLK** | 50% | - |
| $T_{ihrdy}$ | **HREADYIND** hold time after **CLK** | - | 0% |
| $T_{ovrdy}$ | Rising **CLK** to **HREADYOUTD** valid | 30% | - |
| $T_{ohrdy}$ | **HREADYOUTD** hold time from rising **CLK** | >0% | - |
| $T_{ovrsp}$ | Rising **CLK** to **HRESPD** valid | 30% | - |
| $T_{ohrsp}$ | **HRESPD** hold time from rising **CLK** | >0% | - |
| $T_{ovrd}$ | Rising **CLK** to **HRDATAD[31:0]** valid | 30% | - |
| $T_{ohrd}$ | **HRDATAD[31:0]** hold time from rising **CLK** | >0% | - |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.5 Debug interface timing parameters

Figure B-4 shows the debug interface timing parameters.

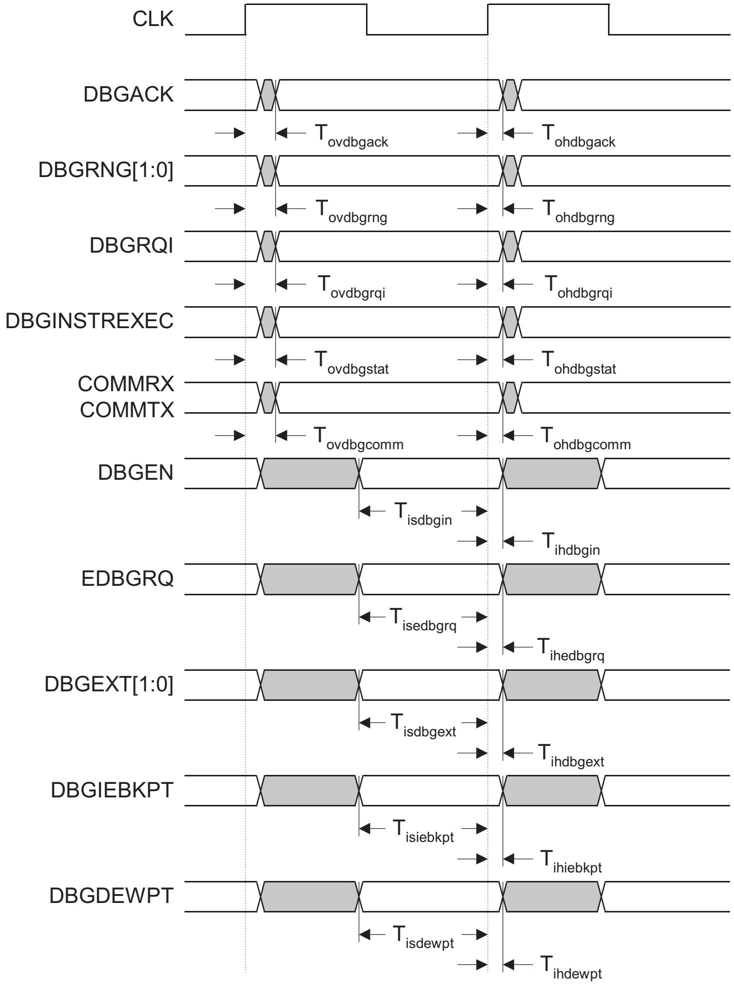


**Figure B-4 Debug interface timing parameters**

Table B-4 on page B-9 describes the timing parameters shown in Figure B-4.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

### Table B-4 Debug interface timing parameters

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovdbgack}$ | Rising **CLK** to **DBGACK** valid | - | 60% |
| $T_{ohdbgack}$ | **DBGACK** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgrng}$ | Rising **CLK** to **DBGRNG[1:0]** valid | - | 80% |
| $T_{ohdbgrng}$ | **DBGRNG[1:0]** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgrqi}$ | Rising **CLK** to **DBGRQI** valid | - | 45% |
| $T_{ohdbgrqi}$ | **DBGRQI** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgstat}$ | Rising **CLK** to **DBGINSTREXEC** valid | - | 45% |
| $T_{ohdbgstat}$ | **DBGINSTREXEC** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgcomm}$ | Rising **CLK** to communications channel outputs valid | - | 60% |
| $T_{ohdbgcomm}$ | Communications channel outputs hold time from rising **CLK** | >0% | - |
| $T_{isdbgen}$ | **DBGEN** input setup time to rising **CLK** | 35% | - |
| $T_{ihdbgen}$ | **DBGEN** input hold time from rising **CLK** | - | 0% |
| $T_{isedbgrq}$ | **EDBGRQ** input setup hold time to rising **CLK** | 30% | - |
| $T_{ihedbgrq}$ | **EDBGRQ** input hold time from rising **CLK** | - | 0% |
| $T_{isdbgext}$ | **DBGEXT** input setup time to rising **CLK** | 20% | |
| $T_{ihdbgext}$ | **DBGEXT** input hold time from rising **CLK** | - | 0% |
| $T_{isiebkpt}$ | **DBGIEBKPT** input setup time to rising **CLK** | 50% | - |
| $T_{ihiebkpt}$ | **DBGIEBKPT** input hold time from rising **CLK** | - | 0% |
| $T_{isdewpt}$ | **DBGDEWPT** input setup time to rising **CLK** | 50% | - |
| $T_{ihdewpt}$ | **DBGDEWPT** input hold time from rising **CLK** | - | 0% |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

## B.6 JTAG interface timing parameters

Figure B-5 shows the JTAG interface timing parameters.

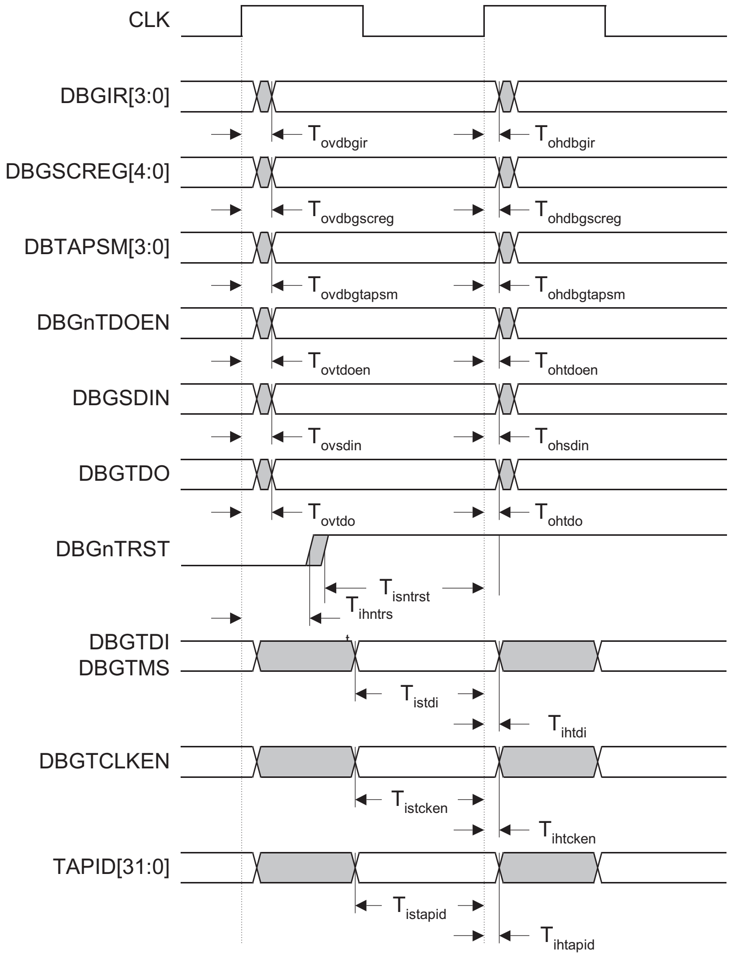


**Figure B-5 JTAG interface timing parameters**

Copyright © 2004, 2006 ARM Limited. All rights reserved.

Table B-5 describes the timing parameters shown in Figure B-5 on page B-10.

**Table B-5 JTAG interface timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovdbgir}$ | Rising **CLK** to **DBGIR** valid | - | 25% |
| $T_{ohdbgir}$ | **DBGIR** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgscreg}$ | Rising **CLK** to **DBGSCREG** valid | - | 30% |
| $T_{ohdbgscreg}$ | **DBGSCREG** hold time from rising **CLK** | >0% | - |
| $T_{ovdbgtapsm}$ | Rising **CLK** to **DBGTAPSM** valid | - | 30% |
| $T_{ohdbgtapsm}$ | **DBGTAPSM** hold time from rising **CLK** | >0% | - |
| $T_{ovtdoen}$ | Rising **CLK** to **DBGnTDOEN** valid | - | 40% |
| $T_{ohtdoen}$ | **DBGnTDOEN** hold time from rising **CLK** | >0% | - |
| $T_{ovsdin}$ | Rising **CLK** to **DBGSDIN** valid | - | 25% |
| $T_{ohsdin}$ | **DBGSDIN** hold time from rising **CLK** | >0% | - |
| $T_{ovtdo}$ | Rising **CLK** to **DBGTDO** valid | - | 65% |
| $T_{ohtdo}$ | **DBGTDO** hold time from rising **CLK** | >0% | - |
| $T_{isntrst}$ | **DBGnTRST** deasserted input setup time to rising **CLK** | 25% | - |
| $T_{ihntrst}$ | **DBGnTRST** input hold time from rising **CLK** | - | 0% |
| $T_{istdi}$ | TAP state control input setup time to rising **CLK** | 30% | - |
| $T_{ihtdi}$ | TAP state control input hold time from rising **CLK** | - | 0% |
| $T_{istcken}$ | **DBGTCKEN** input setup time to rising **CLK** | 50% | - |
| $T_{ihtcken}$ | **DBGTCKEN** input hold time from rising **CLK** | - | 0% |
| $T_{istapid}$ | **TAPID[31:0]** input setup time to rising **CLK** | 35% | - |
| $T_{ihtapid}$ | **TAPID[31:0]** input hold time from rising **CLK** | - | 0% |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.7 Configuration and exception timing parameters

Figure B-6 shows the configuration and exception timing parameters.

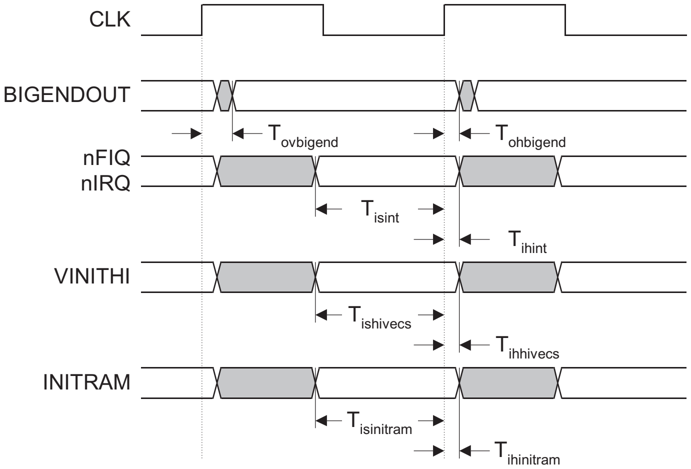


**Figure B-6 Configuration and exception timing parameters**

Table B-6 describes the timing parameters shown in Figure B-6.

**Table B-6 Configuration and exception timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovbigend}$ | Rising **CLK** to **BIGENDOUT** valid | - | 30% |
| $T_{ohbigend}$ | **BIGENDOUT** hold time from rising **CLK** | >0% | - |
| $T_{isint}$ | Interrupt input setup time to rising **CLK** | 30% | - |
| $T_{ihint}$ | Interrupt input hold time from rising **CLK** | - | 0% |
| $T_{ishivecs}$ | **VINITHI** input setup time to rising **CLK** | 90% | - |
| $T_{ihhivecs}$ | **VINITHI** input hold time from rising **CLK** | - | 0% |
| $T_{isinitram}$ | **INITRAM** input setup time to rising **CLK** | 95% | - |
| $T_{ihinitram}$ | **INITRAM** input hold time from rising **CLK** | - | 0% |

The **VINITHI** and **INITRAM** pins are specified as 95% of the cycle because they are for input configuration during reset and can be considered static.

## B.8 INTEST wrapper timing parameters

Figure B-7 shows the INTEST wrapper timing parameters. The INTEST wrapper inputs and outputs are specified as 95% of the cycle because they are production test related and expected to operate at typically 50% of the functional clock rate.

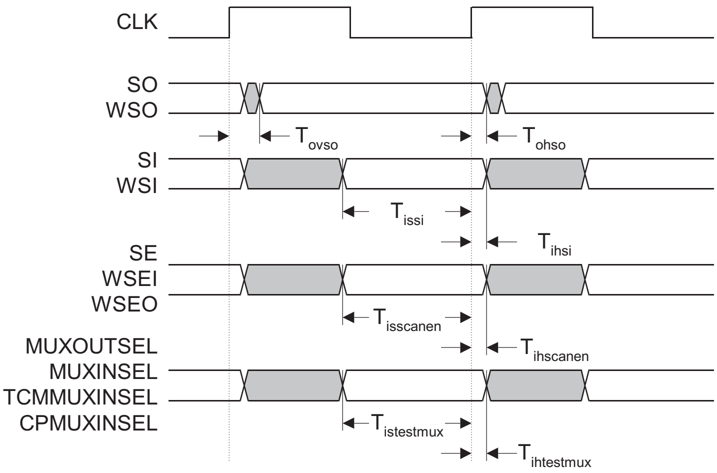


**Figure B-7 INTEST wrapper timing parameters**

Table B-7 describes the timing parameters shown in Figure B-7.

**Table B-7 INTEST wrapper timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovso}$ | Rising **CLK** to **SO** valid | - | 30% |
| $T_{ohso}$ | **SO** hold time from rising **CLK** | >0% | - |
| $T_{issi}$ | **SI** input setup time to rising **CLK** | 95% | - |
| $T_{ihsi}$ | **SI** input hold time from rising **CLK** | - | 0% |
| $T_{isscanen}$ | **SCANEN** input setup time to rising **CLK** | 95% | - |
| $T_{ihscanen}$ | **SCANEN** input hold time from rising **CLK** | - | 0% |
| $T_{istestmux}$ | Test mux input setup time to rising **CLK** | 95% | - |
| $T_{ihtestmux}$ | Test mux input hold time from rising **CLK** | - | 0% |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.9 ETM interface timing parameters

Figure B-8 shows the ETM interface timing parameters.

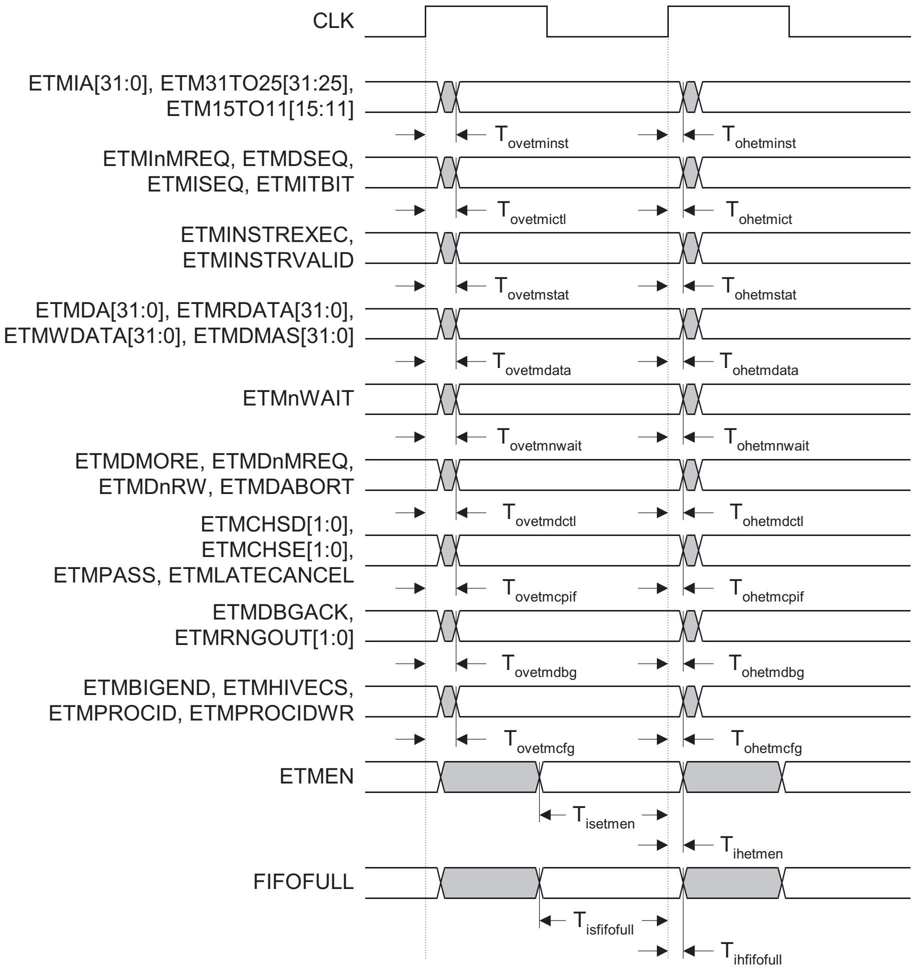


**Figure B-8 ETM interface timing parameters**

Table B-8 describes the timing parameters shown in Figure B-8 on page B-14.

**Table B-8 ETM interface timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovetminst}$ | Rising **CLK** to ETM instruction interface valid | - | 30% |
| $T_{ohetminst}$ | ETM instruction interface hold time from rising **CLK** | >0% | - |
| $T_{ovetmictl}$ | Rising **CLK** to ETM instruction control valid | - | 30% |
| $T_{ohetmictl}$ | ETM instruction control hold time from rising **CLK** | >0% | - |
| $T_{ovetmstat}$ | Rising **CLK** to **ETMINSTREXEC** valid | - | 30% |
| $T_{ohetmstat}$ | **ETMINSTREXEC** hold time from rising **CLK** | >0% | - |
| $T_{ovetmdata}$ | Rising **CLK** to ETM data interface valid | - | 30% |
| $T_{ohetmdata}$ | ETM data interface hold time from rising **CLK** | >0% | - |
| $T_{ovetmnwait}$ | Rising **CLK** to **ETMnWAIT** valid | - | 30% |
| $T_{ohetmnwait}$ | **ETMnWAIT** hold time from rising **CLK** | >0% | - |
| $T_{ovetmdctl}$ | Rising **CLK** to ETM data control valid | - | 30% |
| $T_{ohetmdctl}$ | ETM data control hold time from rising **CLK** | >0% | - |
| $T_{ovetmcfg}$ | Rising **CLK** to ETM configuration valid | - | 30% |
| $T_{ohetmcfg}$ | ETM configuration hold time from rising **CLK** | >0% | - |
| $T_{ovetmcpif}$ | Rising **CLK** to ETM coprocessor signals valid | - | 30% |
| $T_{ohetmcpif}$ | ETM coprocessor signals hold time from rising **CLK** | >0% | - |
| $T_{ovetmdbg}$ | Rising **CLK** to ETM debug signals valid | - | 30% |
| $T_{ohetmdbg}$ | ETM debug signals hold time from rising **CLK** | >0% | - |
| $T_{isetmen}$ | **ETMEN** input setup time to rising **CLK** | 50% | - |
| $T_{ihetmen}$ | **ETMEN** input hold time from rising **CLK** | - | 0% |
| $T_{isfifofull}$ | **FIFOFULL** input setup time to rising **CLK** | 50% | - |
| $T_{ihfifofull}$ | **FIFOFULL** input hold time from rising **CLK** | - | 0% |

Copyright © 2004, 2006 ARM Limited. All rights reserved.

## B.10 TCM interface timing parameters

Figure B-9 shows TCM interface timing parameters.

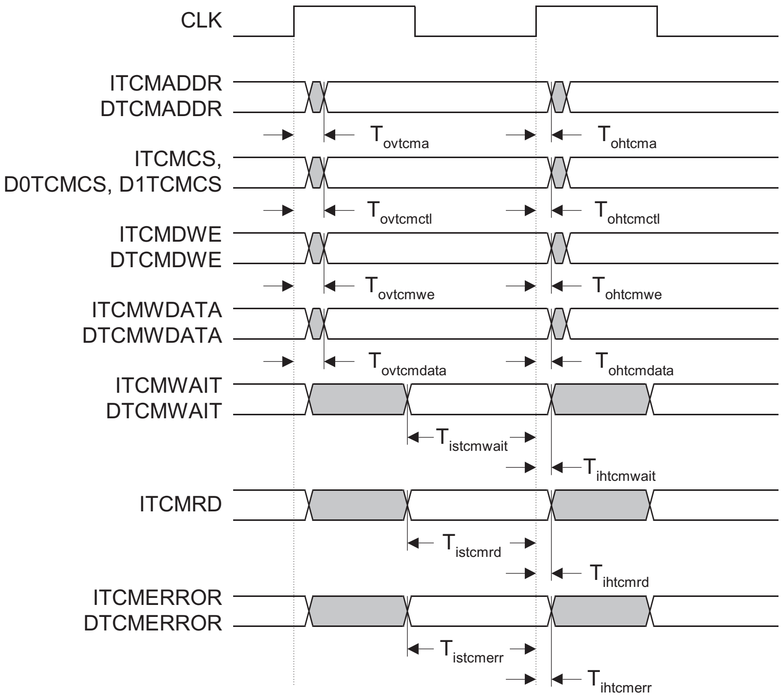


**Figure B-9 TCM interface timing parameters**

Table B-9 describes the timing parameters shown in Figure B-9.

**Table B-9 TCM interface timing parameters**

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovtcma}$ | Rising **CLK** to **TCMADDR** valid | - | 85% |
| $T_{ohtcma}$ | **TCMADDR** hold time from rising **CLK** | >0% | - |
| $T_{ovtcmctl}$ | Rising **CLK** to TCM data control valid | - | 85% |
| $T_{ohtcmctl}$ | TCM data control valid hold time from rising **CLK** | >0% | - |
| $T_{ovtcmwe}$ | Rising **CLK** to TCM write enable valid | - | 85% |
| $T_{ohtcmwe}$ | TCM write enable hold time from rising **CLK** | >0% | - |

Table B-9 TCM interface timing parameters (continued)

| Symbol | Parameter | Min | Max |
|---|---|---|---|
| $T_{ovtcmdata}$ | Rising **CLK** to TCM write data valid | - | 50% |
| $T_{ohtcmdata}$ | TCM write data hold time from rising **CLK** | >0% | - |
| $T_{istcmwait}$ | **TCMWAIT** input setup time to rising **CLK** | 15% | - |
| $T_{ihtcmwait}$ | **TCMWAIT** input hold time from rising **CLK** | - | 0% |
| $T_{istcmrd}$ | TCM read data input setup time to rising **CLK** | 40% | - |
| $T_{ihtcmrd}$ | TCM read data input hold time from rising **CLK** | - | 0% |
| $T_{istcmerr}$ | **TCMERROR** input setup time to rising **CLK** | 40% | - |
| $T_{ihtcmerr}$ | **TCMERROR** input hold time from rising **CLK** | - | 0% |

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

# Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

**Abort**

A mechanism that indicates to a core that it must halt execution of an attempted illegal memory access. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch or Data Abort, and an internal or External Abort.

*See also* Data Abort, External Abort and Prefetch Abort.

**Addressing modes**

A mechanism, shared by many different instructions, for generating values used by the instructions. For four of the ARM addressing modes, the values generated are memory addresses (which is the traditional role of an addressing mode). A fifth addressing mode generates values to be used as operands by data-processing instructions.

**Advanced High-performance Bus (AHB)**

The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

*See also* Advanced Microcontroller Bus Architecture and AHB-Lite.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Advanced Microcontroller Bus Architecture (AMBA)**

AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

**Advanced Peripheral Bus (APB)**

The AMBA Advanced Peripheral Bus is a simpler bus protocol than AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

*See also* Advanced High-performance Bus.

**AHB**

*See* Advanced High-performance Bus.

**AHB-AP**

*See* AHB Access Port.

**AHB-Lite**

AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.

**Aligned**

Aligned data items are stored so that their address is divisible by the highest power of two that divides their size. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively. Other related terms are defined similarly.

**AMBA**

*See* Advanced Microcontroller Bus Architecture.

**APB**

*See* Advanced Peripheral Bus.

**Application Specific Integrated Circuit (ASIC)**

An integrated circuit that has been designed to perform a specific application function. It can be custom-built or mass-produced.

**Architecture**

The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.

**ARM instruction**

A word that specifies an operation for an ARM processor to perform. ARM instructions must be word-aligned.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**ARM state**

A processor that is executing ARM (32-bit) word-aligned instructions is operating in ARM state.

**ASIC**

*See* Application Specific Integrated Circuit.

**ATPG**

*See* Automatic Test Pattern Generation.

**Automatic Test Pattern Generation (ATPG)**

The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

**Banked registers**

Those physical registers whose use is defined by the current processor mode. The banked registers are r8 to r14.

**Base register write-back**

Updating the contents of the base register used in an instruction target address calculation so that the modified address is changed to the next higher or lower sequential address in memory. This means that it is not necessary to fetch the target address for successive instruction transfers and enables faster burst accesses to sequential memory.

**Beat**

Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

*See also* Burst.

**Big-endian**

Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

*See also* Little-endian and Endianness.

**Big-endian memory**

Memory in which:

- a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

*See also* Little-endian memory.

**Boundary scan chain**

A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Breakpoint**

A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.

*See also* Watchpoint.

**Burst**

A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AHB buses are controlled using the **HBURST** signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented.

*See also* Beat.

**Byte**

An 8-bit data item.

**Clock gating**

Gating a clock signal for a macrocell with a control signal and using the modified clock that results to control the operating state of the macrocell.

**Cold reset**

Also known as power-on reset. Starting the processor by turning power on. Turning power off and then back on again clears main memory and many internal settings. Some program failures can lock up the processor and require a cold reset to enable the system to be used again. In other cases, only a warm reset is required.

*See also* Warm reset.

**Communications channel**

The hardware used for communicating between the software running on the processor, and an external host, using the debug interface. When this communication is for debug purposes, it is called the Debug Comms Channel. In an ARMv6 compliant core, the communications channel includes the Data Transfer Register, some bits of the Data Status and Control Register, and the external debug interface controller, such as the DBGTAP controller in the case of the JTAG interface.

**Condition field**

A four-bit field in an instruction that specifies a condition under which the instruction can execute.

**Conditional execution**

If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Control bits**
The bottom eight bits of a Program Status Register (PSR). The control bits change when an exception arises and can be altered by software only when the processor is in a privileged mode.

**Coprocessor**
A processor that supplements the main processor. It carries out additional functions that the main processor cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.

**Core**
A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

**Core reset**
*See* Warm reset.

**CPSR**
*See* Current Program Status Register

**Current Program Status Register (CPSR)**
The register that holds the current operating processor status.

**Data Abort**
An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Data Abort is attempting to access invalid data memory.

*See also* Abort, External Abort, and Prefetch Abort.

**DBGTAP**
*See* Debug Test Access Port.

**Debugger**
A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

**Debug Test Access Port (DBGTAP)**
The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **DBGTDI**, **DBGTDO**, **DBGTMS**, and **TCK**. The optional terminal is **TRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.

**Direct Memory Access (DMA)**
An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

**DMA**
*See* Direct Memory Access.

**DNM**
*See* Do Not Modify.

**Do Not Modify (DNM)**
In Do Not Modify fields, the value must not be altered by software. DNM fields read as Unpredictable values, and must only be written with the same value read from the same field on the same processor.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

DNM fields are sometimes followed by RAZ or RAO in parentheses to show which way the bits should read for future compatibility, but programmers must not rely on this behavior.

**Doubleword** A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

**Doubleword-aligned**

A data item having a memory address that is divisible by eight.

**DSM** *See* Design Simulation Model.

**EmbeddedICE logic** An on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.

**EmbeddedICE-RT** The JTAG-based hardware provided by debuggable ARM processors to aid debugging in real-time.

**Embedded Trace Macrocell (ETM)**

A hardware macrocell that, when connected to a processor core, outputs instruction and data trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol.

**Endianness** Byte ordering. The scheme that determines the order in which successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

*See also* Little-endian and Big-endian

**ETM** See *Embedded Trace Macrocell.*

**Exception** A fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs, normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt handler to deal with the exception.

**Exceptional state** When a potentially exceptional instruction is issued, the VFP11 coprocessor sets the EX bit, FPEXC[31], and loads a copy of the potentially exceptional instruction in the FPINST register. If the instruction is a short vector operation, the register fields in FPINST are altered to point to the potentially exceptional iteration. When in the exceptional state, the issue of a trigger instruction to the VFP11 coprocessor causes a bounce.

*See also* Bounce, Potentially exceptional instruction, and Trigger instruction.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Exception service routine**

*See* Interrupt handler.

**Exception vector**

*See* Interrupt vector.

**Exponent**

The component of a floating-point number that normally signifies the integer power to which two is raised in determining the value of the represented number.

**External Abort**

An indication from an external memory system to a core that it must halt execution of an attempted illegal memory access. An External Abort is caused by the external memory system as a result of attempting to access invalid memory.

*See also* Abort, Data Abort and Prefetch Abort.

**Halfword**

A 16-bit data item.

**Halt mode**

One of two mutually exclusive debug modes. In halt mode all processor execution halts when a breakpoint or watchpoint is encountered. All processor state, coprocessor state, memory and input/output locations can be examined and altered by the JTAG interface.

*See also* Monitor debug-mode.

**High vectors**

Alternative locations for exception vectors. The high vector address range is near the top of the address space, rather than at the bottom.

**IMB**

*See* Instruction Memory Barrier.

**Index register**

A register specified in some load or store instructions. The value of this register is used as an offset to be added to or subtracted from the base register value to form the virtual address, which is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

**Instruction cycle count**

The number of cycles for which an instruction occupies the Execute stage of the pipeline.

**Instruction Memory Barrier (IMB)**

An operation to ensure that the prefetch buffer is flushed of all out-of-date instructions.

**Internal scan chain**

A series of registers connected together to form a path through a device, used during production testing to import test patterns into internal nodes of the device and export the resulting values.

**Interrupt handler**

A program that control of the processor is passed to when an interrupt occurs.

**Interrupt vector**

One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Joint Test Action Group (JTAG)**

The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.

**JTAG** *See* Joint Test Action Group.

**JTAG Access Port (JTAG-AP)**

An optional component of the DAP that provides JTAG access to on-chip components, operating as a JTAG master port to drive JTAG chains throughout a SoC.

**Little-endian** Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

*See also* Big-endian and Endianness.

**Little-endian memory**

Memory in which:

- a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

*See also* Big-endian memory.

**Load/store architecture**

A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

**Load Store Unit (LSU)**

The part of a processor that handles load and store transfers.

**LSU** *See* Load Store Unit.

**Macrocell** A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.

**Microprocessor** *See* Processor.

**Monitor debug-mode**

One of two mutually exclusive debug modes. In Monitor debug-mode the processor enables a software abort handler provided by the debug monitor or operating system debug task. When a breakpoint or watchpoint is encountered, this enables vital system interrupts to continue to be serviced while normal program execution is suspended.

*See also* Halt mode.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Multi-ICE** A JTAG-based tool for debugging embedded systems.

**Penalty** The number of cycles in which no useful Execute stage pipeline activity can occur because an instruction flow is different from that assumed or predicted.

**Power-on reset** *See* Cold reset.

**Prefetching** In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.

**Prefetch Abort** An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Prefetch Abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.

*See also* Data Abort, External Abort and Abort.

**Processor** A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

**Read** Reads are defined as memory operations that have the semantics of a load. That is, the ARM instructions LDM, LDRD, LDC, LDR, LDRT, LDRSH, LDRH, LDRSB, LDRB, LDRBT, LDREX, RFE, STREX, SWP, and SWPB, and the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP. Java instructions that are accelerated by hardware can cause a number of reads to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

**Region** A partition of instruction or data memory space.

**Reserved** A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

**Saved Program Status Register (SPSR)**
The register that holds the CPSR of the task immediately before the exception occurred that caused the switch to the current mode.

**SBO** *See* Should Be One.

**SBZ** *See* Should Be Zero.

**SBZP** *See* Should Be Zero or Preserved.

Copyright © 2004, 2006 ARM Limited. All rights reserved.

**Scalar operation** A VFP coprocessor operation involving a single source register and a single destination register.

*See also* Vector operation.

**Scan chain** A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

**SCREG** The currently selected scan chain number in an ARM TAP controller.

**Should Be One (SBO)**

Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.

**Should Be Zero (SBZ)**

Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.

**Should Be Zero or Preserved (SBZP)**

Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

**Significand** The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of the implied binary point and a fraction field to the right.

**SPSR** *See* Saved Program Status Register

**TAP** *See* Test access port.

**TCM** *See* Tightly coupled memory.

**Test Access Port (TAP)**

The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.

**Thumb instruction** A halfword that specifies an operation for an ARM processor in Thumb state to perform. Thumb instructions must be halfword-aligned.

**Thumb state** A processor that is executing Thumb (16-bit) halfword aligned instructions is operating in Thumb state.

*Copyright © 2004, 2006 ARM Limited. All rights reserved.*

**Tightly coupled memory (TCM)**

An area of low latency memory that provides predictable instruction execution or data load timing in cases where deterministic performance is required. TCMs are suited to holding:

- critical routines (such as for interrupt handling)
- scratchpad data
- data types whose locality is not suited to caching
- critical data structures (such as interrupt stacks).

**Trap**
An exceptional condition in a VFP coprocessor that has the respective exception enable bit set in the FPSCR register. The user trap handler is executed.

**Undefined**
Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architecture Reference Manual* for more details on ARM exceptions.

**UNP**
*See* Unpredictable.

**Unpredictable**
For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

**Unsupported values**

Specific data values that are not processed by the VFP coprocessor hardware but bounced to the support code for completion. These data can include infinities, NaNs, subnormal values, and zeros. An implementation is free to select which of these values is supported in hardware fully or partially, or requires assistance from support code to complete the operation. Any exception resulting from processing unsupported data is trapped to user code if the corresponding exception enable bit for the exception is set.

**Warm reset**
Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

**Watchpoint**
A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

**Word**
A 32-bit data item.

**Write**
Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH. Java

Copyright © 2004, 2006 ARM Limited. All rights reserved.

instructions that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

**Write completion**

The memory system indicates to the processor that a write has been completed at a point in the transaction where the memory system is able to guarantee that the effect of the write is visible to all processors in the system. This is not the case if the write is associated with a memory synchronization primitive, or is to a Device or Strongly Ordered region. In these cases the memory system might only indicate completion of the write when the access has affected the state of the target, unless it is impossible to distinguish between having the effect of the write visible and having the state of target updated.

This stricter requirement for some types of memory ensures that any side-effects of the memory access can be guaranteed by the processor to have taken place. You can use this to prevent the starting of a subsequent operation in the program order until the side-effects are visible.

Copyright © 2004, 2006 ARM Limited. All rights reserved.