

Cortex[™]-M3

r2p0

Technical Reference Manual



Cortex-M3

Technical Reference Manual

Copyright © 2005-2008 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
15 December 2005	A	Confidential	First Release
13 January 2006	B	Non-Confidential	Confidentiality status amended
10 May 2006	C	Non-Confidential	First Release for r1p0
27 September 2006	D	Non-Confidential	First Release for r1p1
13 June 2007	E	Non-Confidential	Minor update with no technical changes
11 April 2008	F	Confidential	Limited release for SC300 r0p0
26 June 2008	G	Non-Confidential	First Release for r2p0

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 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.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is Final (information on a developed product).

Web Address

<http://www.arm.com>

Contents

Cortex-M3 Technical Reference Manual

Preface

About this book	xx
Feedback	xxv

Chapter 1

Introduction

1.1	About the processor	1-2
1.2	Components, hierarchy, and implementation	1-4
1.3	Execution pipeline stages	1-12
1.4	Prefetch Unit	1-14
1.5	Branch target forwarding	1-15
1.6	Store buffers	1-18
1.7	Product revisions	1-19

Chapter 2

Programmer's Model

2.1	About the programmer's model	2-2
2.2	Privileged access and user access	2-3
2.3	Registers	2-4
2.4	Data types	2-10
2.5	Memory formats	2-11
2.6	Instruction set summary	2-13

Chapter 3	System Control	
	3.1 Summary of processor registers	3-2
Chapter 4	Memory Map	
	4.1 About the memory map	4-2
	4.2 Bit-banding	4-5
	4.3 ROM memory table	4-7
Chapter 5	Exceptions	
	5.1 About the exception model	5-2
	5.2 Exception types	5-4
	5.3 Exception priority	5-6
	5.4 Privilege and stacks	5-9
	5.5 Pre-emption	5-11
	5.6 Tail-chaining	5-14
	5.7 Late-arriving	5-15
	5.8 Exit	5-17
	5.9 Resets	5-20
	5.10 Exception control transfer	5-24
	5.11 Setting up multiple stacks	5-25
	5.12 Abort model	5-27
	5.13 Activation levels	5-32
	5.14 Flowcharts	5-34
Chapter 6	Clocking and Resets	
	6.1 Clocking	6-2
	6.2 Resets	6-4
	6.3 Cortex-M3 reset modes	6-5
Chapter 7	Power Management	
	7.1 About power management	7-2
	7.2 System power management	7-3
Chapter 8	Nested Vectored Interrupt Controller	
	8.1 About the NVIC	8-2
	8.2 NVIC programmer's model	8-3
	8.3 Level versus pulse interrupts	8-43
Chapter 9	Memory Protection Unit	
	9.1 About the MPU	9-2
	9.2 MPU programmer's model	9-3
	9.3 MPU access permissions	9-13
	9.4 MPU aborts	9-15
	9.5 Updating an MPU region	9-16
	9.6 Interrupts and updating the MPU	9-19

Chapter 10	Core Debug	
10.1	About core debug	10-2
10.2	Core debug registers	10-3
10.3	Core debug access example	10-12
10.4	Using application registers in core debug	10-13
 Chapter 11	 System Debug	
11.1	About system debug	11-2
11.2	System debug access	11-3
11.3	System debug programmer's model	11-5
11.4	FPB	11-6
11.5	DWT	11-13
11.6	ITM	11-30
11.7	AHB-AP	11-39
 Chapter 12	 Bus Interface	
12.1	About bus interfaces	12-2
12.2	AMBA 3 compliance	12-3
12.3	ICode bus interface	12-4
12.4	DCode bus interface	12-6
12.5	System interface	12-7
12.6	Unifying the code buses	12-9
12.7	External private peripheral interface	12-10
12.8	Access alignment	12-11
12.9	Unaligned accesses that cross regions	12-12
12.10	Bit-band accesses	12-13
12.11	Write buffer	12-14
12.12	Memory attributes	12-15
12.13	AHB timing characteristics	12-16
 Chapter 13	 Debug Port	
13.1	About the DP	13-2
 Chapter 14	 Embedded Trace Macrocell	
14.1	About the ETM	14-2
14.2	Data tracing	14-7
14.3	ETM resources	14-8
14.4	Trace output	14-11
14.5	ETM architecture	14-12
14.6	ETM programmer's model	14-16
 Chapter 15	 Embedded Trace Macrocell Interface	
15.1	About the ETM interface	15-2
15.2	CPU ETM interface port descriptions	15-3
15.3	Branch status interface	15-6

Chapter 16	AHB Trace Macrocell Interface	
16.1	About the AHB trace macrocell interface	16-2
16.2	CPU AHB trace macrocell interface port descriptions	16-3
Chapter 17	Trace Port Interface Unit	
17.1	About the TPIU	17-2
17.2	TPIU registers	17-8
17.3	Serial wire output connection	17-21
Chapter 18	Instruction Timing	
18.1	About instruction timing	18-2
18.2	Processor instruction timings	18-3
18.3	Load-store timings	18-7
Chapter 19	AC Characteristics	
19.1	Processor timing parameters	19-2
Appendix A	Signal Descriptions	
A.1	Clocks	A-2
A.2	Resets	A-3
A.3	Miscellaneous	A-4
A.4	Interrupt interface	A-6
A.5	Low power interface	A-7
A.6	ICode interface	A-8
A.7	DCode interface	A-9
A.8	System bus interface	A-10
A.9	Private Peripheral Bus interface	A-11
A.10	ITM interface	A-12
A.11	AHB-AP interface	A-13
A.12	ETM interface	A-14
A.13	AHB Trace Macrocell interface	A-16
A.14	Test interface	A-17
A.15	WIC interface	A-18
Appendix B	Revisions	
	Glossary	

List of Tables

Cortex-M3 Technical Reference Manual

	Change History	ii
Table 2-1	Application Program Status Register bit assignments	2-6
Table 2-2	Interrupt Program Status Register bit assignments	2-7
Table 2-3	Bit functions of the EPSR	2-8
Table 2-4	16-bit Cortex-M3 instruction summary	2-13
Table 2-5	32-bit Cortex-M3 instruction summary	2-16
Table 3-1	NVIC registers	3-2
Table 3-2	Core debug registers	3-5
Table 3-3	Flash patch register summary	3-6
Table 3-4	DWT register summary	3-7
Table 3-5	ITM register summary	3-9
Table 3-6	AHB-AP register summary	3-10
Table 3-7	Summary of Debug interface port registers	3-10
Table 3-8	MPU registers	3-11
Table 3-9	TPIU registers	3-12
Table 3-10	ETM registers	3-13
Table 4-1	Memory interfaces	4-3
Table 4-2	Memory region permissions	4-4
Table 4-3	ROM table	4-7
Table 5-1	Exception types	5-4
Table 5-2	Priority-based actions of exceptions	5-6
Table 5-3	Priority grouping	5-8
Table 5-4	Exception entry steps	5-12

Table 5-5	Exception exit steps	5-17
Table 5-6	Exception return behavior	5-19
Table 5-7	Reset actions	5-20
Table 5-8	Reset boot-up behavior	5-21
Table 5-9	Transferring to exception processing	5-24
Table 5-10	Faults	5-28
Table 5-11	Debug faults	5-30
Table 5-12	Fault status and fault address registers	5-31
Table 5-13	Privilege and stack of different activation levels	5-32
Table 5-14	Exception transitions	5-32
Table 5-15	Exception subtype transitions	5-33
Table 6-1	Cortex-M3 processor clocks	6-2
Table 6-2	Cortex-M3 macrocell clocks	6-2
Table 6-3	Reset inputs	6-4
Table 6-4	Reset modes	6-5
Table 7-1	Supported sleep modes	7-3
Table 8-1	NVIC registers	8-3
Table 8-2	Interrupt Controller Type Register bit assignments	8-8
Table 8-3	Auxiliary Control Register bit assignments	8-9
Table 8-4	SysTick Control and Status Register bit assignments	8-10
Table 8-5	SysTick Reload Value Register bit assignments	8-11
Table 8-6	SysTick Current Value Register bit assignments	8-12
Table 8-7	SysTick Calibration Value Register bit assignments	8-12
Table 8-8	Interrupt Set-Enable Register bit assignments	8-14
Table 8-9	Interrupt Clear-Enable Register bit assignments	8-14
Table 8-10	Interrupt Set-Pending Register bit assignments	8-15
Table 8-11	Interrupt Clear-Pending Registers bit assignments	8-16
Table 8-12	Active Bit Register bit assignments	8-16
Table 8-13	Interrupt Priority Registers 0-31 bit assignments	8-18
Table 8-14	CPUID Base Register bit assignments	8-18
Table 8-15	Interrupt Control State Register bit assignments	8-20
Table 8-16	Vector Table Offset Register bit assignments	8-22
Table 8-17	Application Interrupt and Reset Control Register bit assignments	8-23
Table 8-18	System Control Register bit assignments	8-26
Table 8-19	Configuration Control Register bit assignments	8-27
Table 8-20	System Handler Priority Registers bit assignments	8-29
Table 8-21	System Handler Control and State Register bit assignments	8-30
Table 8-22	Memory Manage Fault Status Register bit assignments	8-33
Table 8-23	Bus Fault Status Register bit assignments	8-35
Table 8-24	Usage Fault Status Register bit assignments	8-36
Table 8-25	Hard Fault Status Register bit assignments	8-38
Table 8-26	Debug Fault Status Register bit assignments	8-39
Table 8-27	Memory Manage Fault Address Register bit assignments	8-40
Table 8-28	Bus Fault Address Register bit assignments	8-41
Table 8-29	Auxiliary Fault Status Register bit assignments	8-42
Table 8-30	Software Trigger Interrupt Register bit assignments	8-42
Table 9-1	MPU registers	9-3

Table 9-2	MPU Type Register bit assignments	9-4
Table 9-3	MPU Control Register bit assignments	9-6
Table 9-4	MPU Region Number Register bit assignments	9-7
Table 9-5	MPU Region Base Address Register bit assignments	9-8
Table 9-6	MPU Region Attribute and Size Register bit assignments	9-9
Table 9-7	MPU protection region size field	9-10
Table 9-8	TEX, C, B encoding	9-13
Table 9-9	Cache policy for memory attribute encoding	9-14
Table 9-10	AP encoding	9-14
Table 9-11	XN encoding	9-14
Table 10-1	Core debug registers	10-2
Table 10-2	Debug Halting Control and Status Register	10-4
Table 10-3	Debug Core Register Selector Register	10-7
Table 10-4	Debug Exception and Monitor Control Register	10-9
Table 10-5	Application registers for use in core debug	10-13
Table 11-1	FPB register summary	11-7
Table 11-2	Flash Patch Control Register bit assignments	11-8
Table 11-3	COMP mapping	11-10
Table 11-4	Flash Patch Remap Register bit assignments	11-11
Table 11-5	Flash Patch Comparator Registers bit assignments	11-12
Table 11-6	DWT register summary	11-14
Table 11-7	DWT Control Register bit assignments	11-16
Table 11-8	DWT Current PC Sampler Cycle Count Register bit assignments	11-19
Table 11-9	DWT CPI Count Register bit assignments	11-20
Table 11-10	DWT Exception Overhead Count Register bit assignments	11-21
Table 11-11	DWT Sleep Count Register bit assignments	11-22
Table 11-12	DWT LSU Count Register bit assignments	11-23
Table 11-13	DWT Fold Count Register bit assignments	11-23
Table 11-14	DWT Program Counter Sample Register bit assignments	11-24
Table 11-15	DWT Comparator Registers 0-3 bit assignments	11-24
Table 11-16	DWT Mask Registers 0-3 bit assignments	11-25
Table 11-17	Bit functions of DWT Function Registers 0-3	11-26
Table 11-18	Settings for DWT Function Registers	11-28
Table 11-19	ITM register summary	11-30
Table 11-20	ITM Trace Enable Register bit assignments	11-32
Table 11-21	ITM Trace Privilege Register bit assignments	11-33
Table 11-22	ITM Trace Control Register bit assignments	11-34
Table 11-23	ITM Integration Write Register bit assignments	11-36
Table 11-24	ITM Integration Read Register bit assignments	11-36
Table 11-25	ITM Integration Mode Control Register bit assignments	11-37
Table 11-26	ITM Lock Access Register bit assignments	11-37
Table 11-27	ITM Lock Status Register bit assignments	11-38
Table 11-28	AHB-AP register summary	11-40
Table 11-29	AHB-AP Control and Status Word Register bit assignments	11-41
Table 11-30	AHB-AP Transfer Address Register bit assignments	11-42
Table 11-31	AHB-AP Data Read/Write Register bit assignments	11-43
Table 11-32	AHB-AP Banked Data Register bit assignments	11-43

Table 11-33	AHB-AP Debug ROM Address Register bit assignments	11-44
Table 11-34	AHB-AP ID Register bit assignments	11-44
Table 12-1	Instruction fetches	12-4
Table 12-2	Bus mapper unaligned accesses	12-11
Table 12-3	Memory attributes	12-15
Table 12-4	Interface timing characteristics	12-16
Table 14-1	ETM core interface inputs and outputs	14-4
Table 14-2	Miscellaneous configuration inputs	14-4
Table 14-3	Trace port signals	14-5
Table 14-4	Other signals	14-5
Table 14-5	Clocks and resets	14-6
Table 14-6	APB interface signals	14-6
Table 14-7	Cortex-M3 resources	14-8
Table 14-8	Exception tracing mapping	14-13
Table 14-9	ETM registers	14-16
Table 14-10	Boolean function encoding for events	14-22
Table 14-11	Resource identification encoding	14-23
Table 14-12	Input connections	14-23
Table 14-13	Trigger output connections	14-23
Table 15-1	ETM interface ports	15-3
Table 15-2	Branch status signal function	15-6
Table 15-3	Branches and stages evaluated by the processor	15-7
Table 15-4	Example of an opcode sequence	15-11
Table 16-1	AHB interface ports	16-3
Table 17-1	Trace out port signals	17-5
Table 17-2	ATB port signals	17-6
Table 17-3	Miscellaneous configuration inputs	17-6
Table 17-4	APB interface	17-7
Table 17-5	TPIU registers	17-8
Table 17-6	Async Clock Prescaler Register bit assignments	17-10
Table 17-7	Selected Pin Protocol Register bit assignments	17-11
Table 17-8	Formatter and Flush Status Register bit assignments	17-12
Table 17-9	Formatter and Flush Control Register bit assignments	17-13
Table 17-10	Integration Test Register-ITATBCTR2 bit assignments	17-15
Table 17-11	Integration Test Register-ITATBCTR0 bit assignments	17-16
Table 17-12	Integration Mode Control Register bit assignments	17-17
Table 17-13	Integration Register : TRIGGER bit assignments	17-17
Table 17-14	Integration register : FIFO data 0 bit assignments	17-18
Table 17-15	Integration register : FIFO data 1 bit assignments	17-19
Table 18-1	Instruction timings	18-3
Table 19-1	Miscellaneous input ports timing parameters	19-2
Table 19-2	Low power input ports timing parameters	19-2
Table 19-3	Interrupt input ports timing parameters	19-3
Table 19-4	AHB input ports timing parameters	19-3
Table 19-5	PPB input port timing parameters	19-4
Table 19-6	Debug input ports timing parameters	19-4
Table 19-7	Test input ports timing parameters	19-5

Table 19-8	ETM input port timing parameters	19-5
Table 19-9	Miscellaneous output ports timing parameters	19-5
Table 19-10	Low power output ports timing parameters	19-6
Table 19-11	AHB output ports timing parameters	19-6
Table 19-12	PPB output ports timing parameters	19-8
Table 19-13	Debug interface output ports timing parameters	19-8
Table 19-14	ETM interface output ports timing parameters	19-9
Table 19-15	HTM interface output ports timing parameters	19-9
Table 19-16	Test output ports timing parameters	19-10
Table A-1	Clock signals	A-2
Table A-2	Reset signals	A-3
Table A-3	Miscellaneous signals	A-4
Table A-4	Interrupt interface signals	A-6
Table A-5	Low power interface signals	A-7
Table A-6	ICode interface	A-8
Table A-7	DCode interface	A-9
Table A-8	System bus interface	A-10
Table A-9	Private Peripheral Bus interface	A-11
Table A-10	ITM interface	A-12
Table A-11	AHB-AP interface	A-13
Table A-12	ETM interface	A-14
Table A-13	HTM interface	A-16
Table A-14	Test interface	A-17
Table A-15	WIC interface signals	A-18
Table B-1	Differences between issue E and issue F	B-1
Table B-2	Differences between issue F and issue G	B-5

List of Figures

Cortex-M3 Technical Reference Manual

	Key to timing diagram conventions	xxiii
Figure 1-1	Cortex-M3 block diagram	1-5
Figure 1-2	Cortex-M3 pipeline stages	1-12
Figure 2-1	Processor register set	2-4
Figure 2-2	Application Program Status Register bit assignments	2-6
Figure 2-3	Interrupt Program Status Register bit assignments	2-6
Figure 2-4	Execution Program Status Register	2-8
Figure 2-5	Little-endian and big-endian memory formats	2-12
Figure 4-1	Processor memory map	4-2
Figure 4-2	Bit-band mapping	4-6
Figure 5-1	Stack contents after pre-emption	5-11
Figure 5-2	Exception entry timing	5-13
Figure 5-3	Tail-chaining timing	5-14
Figure 5-4	Late-arriving exception timing	5-15
Figure 5-5	Exception exit timing	5-18
Figure 5-6	Interrupt handling flowchart	5-34
Figure 5-7	Pre-emption flowchart	5-35
Figure 5-8	Return from interrupt flowchart	5-36
Figure 6-1	Reset signals	6-6
Figure 6-2	Power-on reset	6-6
Figure 6-3	Internal reset synchronization	6-7
Figure 7-1	SLEEPING power control example	7-4
Figure 7-2	SLEEPDEEP power control example	7-5

Figure 7-3	WIC mode enable sequence	7-7
Figure 7-4	Power down timing sequence	7-8
Figure 7-5	PMU, WIC, and Cortex-M3 interconnect	7-9
Figure 8-1	Interrupt Controller Type Register bit assignments	8-7
Figure 8-2	Auxiliary Control Register bit assignments	8-8
Figure 8-3	SysTick Control and Status Register bit assignments	8-9
Figure 8-4	SysTick Reload Value Register bit assignments	8-11
Figure 8-5	SysTick Current Value Register bit assignments	8-11
Figure 8-6	SysTick Calibration Value Register bit assignments	8-12
Figure 8-7	Interrupt Priority Registers 0-31 bit assignments	8-17
Figure 8-8	CPUID Base Register bit assignments	8-18
Figure 8-9	Interrupt Control State Register bit assignments	8-20
Figure 8-10	Vector Table Offset Register bit assignments	8-22
Figure 8-11	Application Interrupt and Reset Control Register bit assignments	8-23
Figure 8-12	System Control Register bit assignments	8-25
Figure 8-13	Configuration Control Register bit assignments	8-27
Figure 8-14	System Handler Priority Registers bit assignments	8-29
Figure 8-15	System Handler Control and State Register bit assignments	8-30
Figure 8-16	Configurable Fault Status Registers bit assignments	8-32
Figure 8-17	Memory Manage Fault Status Register bit assignments	8-33
Figure 8-18	Bus Fault Status Register bit assignments	8-34
Figure 8-19	Usage Fault Status Register bit assignments	8-36
Figure 8-20	Hard Fault Status Register bit assignments	8-37
Figure 8-21	Debug Fault Status Register bit assignments	8-39
Figure 8-22	Software Trigger Interrupt Register bit assignments	8-42
Figure 9-1	MPU Type Register bit assignments	9-4
Figure 9-2	MPU Control Register bit assignments	9-5
Figure 9-3	MPU Region Number Register bit assignments	9-7
Figure 9-4	MPU Region Base Address Register bit assignments	9-8
Figure 9-5	MPU Region Attribute and Size Register bit assignments	9-9
Figure 10-1	Debug Halting Control and Status Register bit assignments	10-4
Figure 10-2	Debug Core Register Selector Register bit assignments	10-6
Figure 10-3	Debug Exception and Monitor Control Register bit assignments	10-9
Figure 11-1	System debug access block diagram	11-4
Figure 11-2	Flash Patch Control Register bit assignments	11-8
Figure 11-3	Flash Patch Remap Register bit assignments	11-10
Figure 11-4	Flash Patch Comparator Registers bit assignments	11-11
Figure 11-5	DWT Control Register bit assignments	11-16
Figure 11-6	DWT CPI Count Register bit assignments	11-20
Figure 11-7	DWT Exception Overhead Count Register bit assignments	11-21
Figure 11-8	DWT Sleep Count Register bit assignments	11-21
Figure 11-9	DWT LSU Count Register bit assignments	11-22
Figure 11-10	DWT Fold Count Register bit assignments	11-23
Figure 11-11	DWT Mask Registers 0-3 bit assignments	11-25
Figure 11-12	DWT Function Registers 0-3 bit assignments	11-26
Figure 11-13	ITM Trace Privilege Register bit assignments	11-33
Figure 11-14	ITM Trace Control Register bit assignments	11-34

Figure 11-15	ITM Integration Write Register bit assignments	11-35
Figure 11-16	ITM Integration Read Register bit assignments	11-36
Figure 11-17	ITM Integration Mode Control bit assignments	11-37
Figure 11-18	ITM Lock Status Register bit assignments	11-38
Figure 11-19	AHB-AP Control and Status Word Register	11-41
Figure 11-20	AHB-AP ID Register	11-44
Figure 12-1	ICode/DCode multiplexer	12-9
Figure 14-1	ETM block diagram	14-3
Figure 14-2	Return from exception packet encoding	14-12
Figure 14-3	Exception encoding for branch packet	14-14
Figure 15-1	Conditional branch backwards not taken	15-8
Figure 15-2	Conditional branch backwards taken	15-9
Figure 15-3	Conditional branch forwards not taken	15-9
Figure 15-4	Conditional branch forwards taken	15-9
Figure 15-5	Unconditional branch without pipeline stalls	15-10
Figure 15-6	Unconditional branch with pipeline stalls	15-10
Figure 15-7	Unconditional branch in execute aligned	15-11
Figure 15-8	Unconditional branch in execute unaligned	15-11
Figure 15-9	Example of an opcode sequence	15-13
Figure 17-1	TPIU block diagram (non-ETM version)	17-3
Figure 17-2	TPIU block diagram (ETM version)	17-4
Figure 17-3	Supported Sync Port Size Register bit assignments	17-10
Figure 17-4	Async Clock Prescaler Register bit assignments	17-10
Figure 17-5	Selected Pin Protocol Register bit assignments	17-11
Figure 17-6	Formatter and Flush Status Register bit assignments	17-12
Figure 17-7	Formatter and Flush Control Register bit assignments	17-13
Figure 17-8	Integration Test Register-ITATBCTR2 bit assignments	17-15
Figure 17-9	Integration Test Register-ITATBCTR0 bit assignments	17-16
Figure 17-10	Integration Mode Control Register bit assignments	17-16
Figure 17-11	Integration Register : TRIGGER bit assignments	17-17
Figure 17-12	Integration register : FIFO data 0 bit assignments	17-18
Figure 17-13	Integration register : FIFO data 1 bit assignments	17-19
Figure 17-14	Dedicated pin used for TRACESWO	17-21
Figure 17-15	SWO shared with TRACEPORT	17-22
Figure 17-16	SWO shared with JTAG-TDO	17-22

Preface

This preface introduces the *Cortex-M3 Technical Reference Manual* (TRM). It contains the following sections:

- *About this book* on page xx
- *Feedback* on page xxv.

About this book

This book is for the Cortex-M3 processor.

Product revision status

The *mpn* identifier indicates the revision status of the product described in this manual, where:

rn Identifies the major revision of the product.

pn Identifies the minor revision or modification status of the product.

Intended audience

This manual is written to help system designers, system integrators, and verification engineers who are implementing a *System-on-Chip* (SoC) device based on the Cortex-M3 processor.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for a description of the components of the processor, and about the processor instruction set.

Chapter 2 *Programmer's Model*

Read this for a description of the processor register set, modes of operation, and other information for programming the processor.

Chapter 3 *System Control*

Read this for a description of the registers and programmer's model for system control.

Chapter 4 *Memory Map*

Read this for a description of the processor memory map and bit-banding feature.

Chapter 5 *Exceptions*

Read this for a description of the processor exception model.

Chapter 6 *Clocking and Resets*

Read this chapter for a description of the processor clocking and resets.

Chapter 7 Power Management

Read this for a description of the processor power management and power saving.

Chapter 8 Nested Vectored Interrupt Controller

Read this for a description of the processor interrupt processing and control.

Chapter 9 Memory Protection Unit

Read this for a description of the processor *Memory Protection Unit* (MPU).

Chapter 10 Core Debug

Read this chapter to learn about debugging and testing the processor core.

Chapter 11 System Debug

Read this for a description of the processor system debug components.

Chapter 13 Debug Port

Read this for a description of the processor debug port, and the *Serial Wire JTAG Debug Port* (SWJ-DP) and *Serial Wire Debug Port* (SW-DP).

Chapter 17 Trace Port Interface Unit

Read this chapter to learn about the processor *Trace Port Interface Unit* (TPIU).

Chapter 12 Bus Interface

Read this for a description of the processor bus interfaces.

Chapter 14 Embedded Trace Macrocell

Read this for a description of the processor *Embedded Trace Macrocell* (ETM).

Chapter 15 Embedded Trace Macrocell Interface

Read this for a description of the processor ETM interface.

Chapter 16 AHB Trace Macrocell Interface

Read this for a description of the processor *Advanced High-performance Bus* (AHB) trace macrocell interface.

Chapter 18 Instruction Timing

Read this for a description of the processor instruction timing and clock cycles.

Chapter 19 AC Characteristics

Read this for a description of the processor ac characteristics.

Appendix A Signal Descriptions

Read this for a summary of processor signals.

Appendix B Revisions

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

Glossary Read this for definitions of terms used in this book.

Conventions

Conventions that this book can use are described in:

- *Typographical*
- *Timing diagrams* on page xxiii
- *Signals* on page xxiii.

Typographical

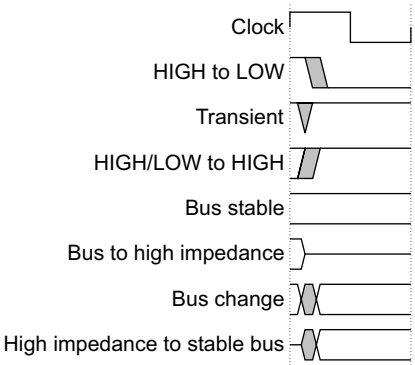
The typographical conventions are:

<i>italic</i>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>monospace</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
monospace	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>

Timing diagrams

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

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



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means: <ul style="list-style-type: none">• HIGH for active-HIGH signals• LOW for active-LOW signals.
Lower-case n	At the start or end of a signal name denotes an active-LOW signal.
Prefix A	Denotes global <i>Advanced eXtensible Interface</i> (AXI) signals.
Prefix AR	Denotes AXI read address channel signals.
Prefix AW	Denotes AXI write address channel signals.
Prefix B	Denotes AXI write response channel signals.
Prefix C	Denotes AXI low-power interface signals.
Prefix H	Denotes <i>Advanced High-performance Bus</i> (AHB) signals.

Prefix P	Denotes <i>Advanced Peripheral Bus</i> (APB) signals.
Prefix R	Denotes AXI read data channel signals.
Prefix W	Denotes AXI write data channel signals.

Further reading

This section lists publications by ARM and by third parties.

See <http://infocenter.arm.com> for access to ARM documentation.

ARM publications

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

- *ARMv7-M Architecture Reference Manual* (ARM DDI 0403)
- *ARM AMBA® 3 AHB-Lite Protocol (v1.0)* (ARM IHI 0033)
- *ARM CoreSight™ Components Technical Reference Manual* (ARM DDI 0314)
- *ARM Debug Interface v5, Architecture Specification* (ARM IHI 0031)
- *ARM Embedded Trace Macrocell Architecture Specification* (ARM IHI 0014).

Other publications

This section lists relevant documents published by third parties:

- IEEE Standard *Test Access Port and Boundary-Scan Architecture* 1149.1-2001 (JTAG).

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on this manual

If you have any comments on this book, send email to errata@arm.com. Give:

- the title
- the number
- the page number(s) to which your comments refer
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1

Introduction

This chapter introduces the processor and instruction set. It contains the following sections:

- *About the processor* on page 1-2
- *Components, hierarchy, and implementation* on page 1-4
- *Execution pipeline stages* on page 1-12
- *Prefetch Unit* on page 1-14
- *Branch target forwarding* on page 1-15
- *Store buffers* on page 1-18
- *Product revisions* on page 1-19.

1.1 About the processor

The processor is a low-power processor that features low gate count, low interrupt latency, and low-cost debug. It is intended for deeply embedded applications that require fast interrupt response features. The processor implements the ARMv7-M architecture.

The processor incorporates:

- Processor core. A low gate count core, with low latency interrupt processing that features:
 - A Thumb instruction set subset, defined in the *ARMv7-M Architecture Reference Manual*.
 - Banked *Stack Pointer* (SP) only.
 - Hardware divide instructions, SDIV and UDIV (Thumb 32-bit instructions).
 - Handler and Thread modes.
 - Thumb and Debug states.
 - Interruptible-continued LDM/STM, PUSH/POP for low interrupt latency.
 - Automatic processor state saving and restoration for low latency *Interrupt Service Routine* (ISR) entry and exit.
 - Support for ARMv6 BE8 or LE accesses.
 - Support for ARMv6 unaligned accesses.
- *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor core to achieve low latency interrupt processing. Features include:
 - External interrupts of 1 to 240 configurable size.
 - Bits of priority of 3 to 8 configurable size.
 - Dynamic reprioritization of interrupts.
 - Priority grouping. This enables selection of pre-empting interrupt levels and non pre-empting interrupt levels.
 - Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
 - Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.
- *Memory Protection Unit* (MPU). An optional MPU for memory protection:
 - Eight memory regions.

- *Sub Region Disable* (SRD), enabling efficient use of memory regions.
- You can enable a background region that implements the default memory map attributes.
- Bus interfaces:
 - *Advanced High-performance Bus-Lite* (AHB-Lite) ICode, DCode and System bus interfaces.
 - *Private Peripheral Bus* (PPB) based on *Advanced Peripheral Bus* (APB) interface.
 - Bit band support that includes atomic bit band write and read operations.
 - Memory access alignment.
 - Write buffer for buffering of write data.
 - Exclusive access transfers for multiprocessor systems.
- Low-cost debug solution that features:
 - Debug access to all memory and registers in the system, including access to memory mapped devices, access to internal core registers when the core is halted, and access to debug control registers even while **SYSRESETn** is asserted.
 - *Serial Wire Debug Port* (SW-DP) or *Serial Wire JTAG Debug Port* (SWJ-DP) debug access, or both.
 - *Flash Patch and Breakpoint* (FPB) unit for implementing breakpoints and code patches.
 - *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.
 - *Instrumentation Trace Macrocell* (ITM) for support of printf style debugging.
 - *Trace Port Interface Unit* (TPIU) for bridging to a *Trace Port Analyzer* (TPA).
 - Optional *Embedded Trace Macrocell* (ETM) for instruction trace.

1.2 Components, hierarchy, and implementation

This section describes the components, hierarchy, and implementation of the processor. It also describes the configurable options. The main blocks are:

- *Processor core* on page 1-5
- *NVIC* on page 1-7
- *Bus matrix* on page 1-7
- *FPB* on page 1-8
- *DWT* on page 1-8
- *ITM* on page 1-8
- *MPU* on page 1-9
- *ETM* on page 1-9
- *AHB-AP* on page 1-9
- *AHB Trace Macrocell interface* on page 1-9
- *TPIU* on page 1-9
- *WIC* on page 1-10
- *SW/SWJ-DP* on page 1-10
- *Interrupts* on page 1-11
- *Observation* on page 1-11
- *ROM table* on page 1-11.

Figure 1-1 on page 1-5 shows the structure of the processor.

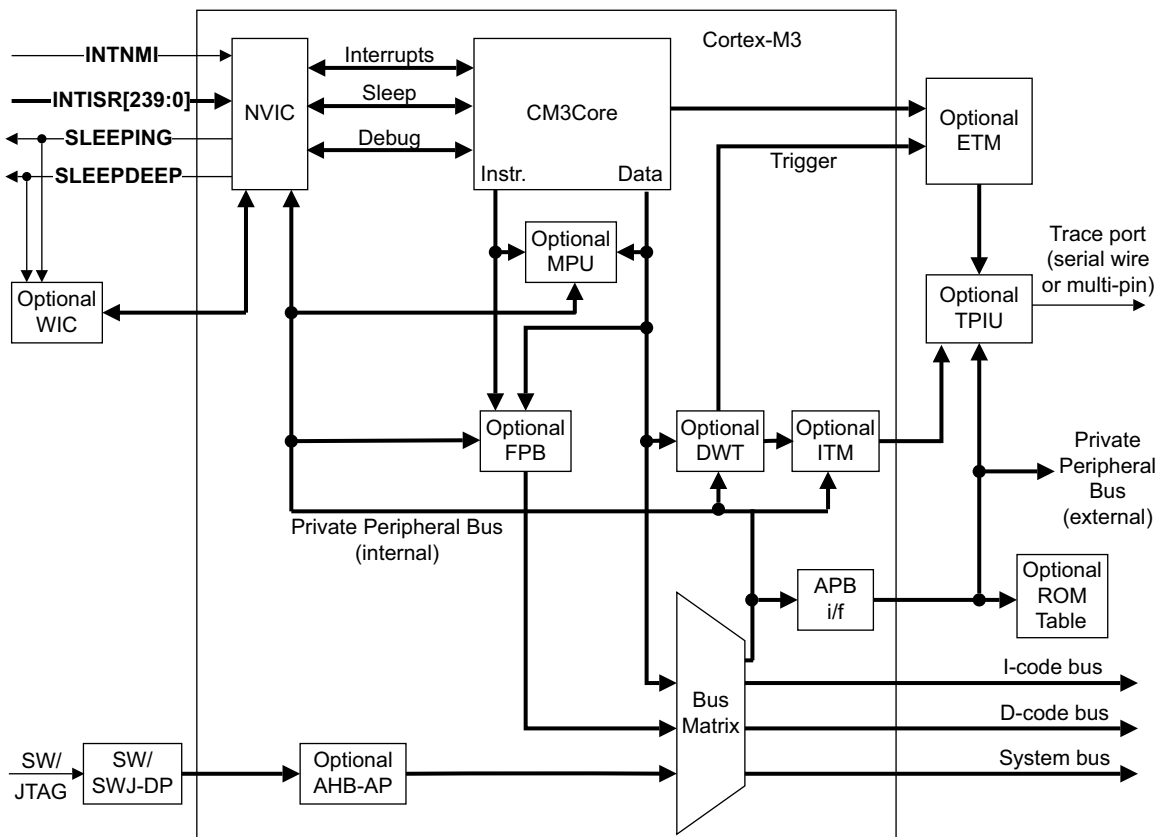


Figure 1-1 Cortex-M3 block diagram

1.2.1 Processor core

The processor core implements the ARMv7-M architecture. It has the following main features:

- Thumb instruction set subset, consisting of all base Thumb instructions, 16-bit and 32-bit. See the *ARMv7-M Architecture Reference Manual* for more information.
- Harvard processor architecture enabling simultaneous instruction fetch with data load/store.
- Three-stage pipeline.
- Single cycle 32-bit multiply.

- Hardware divide.
- Thumb and Debug states.
- Handler and Thread modes.
- Low latency ISR entry and exit.
 - Processor state saving and restoration, with no instruction fetch overhead. Exception vector is fetched from memory in parallel with the state saving, enabling faster ISR entry.
 - Support for late arriving interrupts.
 - Tightly coupled interface to interrupt controller enabling efficient processing of late-arriving interrupts.
 - Tail-chaining of interrupts, enabling back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
- Interruptible-continued LDM/STM, PUSH/POP.
- ARMv6 compatible BE8 and LE access support.
- ARMv6 compatible unaligned access support.

Registers

The processor contains:

- 13 general purpose 32-bit registers, R0 to R12
- *Link Register* (LR)
- *Program Counter* (PC)
- *Program Status Register*, xPSR
- two banked SP registers.

Memory interface

The processor has a Harvard interface to enable simultaneous instruction fetches with data load/stores. Memory accesses are controlled by:

- A separate *Load Store Unit* (LSU) that decouples load and store operations from the *Arithmetic and Logic Unit* (ALU).
- A 3-word entry *Prefetch Unit* (PFU). One word is fetched at a time. This can be two Thumb instructions, one word-aligned Thumb 32-bit instruction, or the upper/lower halfword of a halfword-aligned Thumb 32-bit instruction with one Thumb instruction, or the lower/upper halfword of another halfword-aligned Thumb 32-bit instruction. All fetch addresses from the core are word aligned. If

a Thumb 32-bit instruction is halfword aligned, two fetches are necessary to fetch the Thumb 32-bit instruction. However, the 3-entry prefetch buffer ensures that a stall cycle is only necessary for the first halfword Thumb 32-bit instruction fetched.

1.2.2 NVIC

The NVIC is tightly coupled to the processor core. This facilitates low latency exception processing. The main features include:

- a configurable number of external interrupts, from 1 to 240
- a configurable number of bits of priority, from three to eight bits
- level and pulse interrupt support
- dynamic reprioritization of interrupts
- priority grouping
- support for tail-chaining of interrupts
- processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.

Chapter 8 *Nested Vectored Interrupt Controller* describes the NVIC in detail.

1.2.3 Bus matrix

The bus matrix connects the processor and debug interface to the external buses. The bus matrix interfaces to the following external buses:

- ICode bus. This is for instruction and vector fetches from code space. This is a 32-bit AHB-Lite bus.
- DCode bus. This is for data load/stores and debug accesses to code space. This is a 32-bit AHB-Lite bus.
- System bus. This is for instruction and vector fetches, data load/stores and debug accesses to system space. This is a 32-bit AHB-Lite bus.
- PPB. This is for data load/stores and debug accesses to PPB space. This is a 32-bit APB (v3.0) bus.

The bus matrix also controls the following:

- Unaligned accesses. The bus matrix converts unaligned processor accesses into aligned accesses.

- Bit-banding. The bus matrix converts bit-band alias accesses into bit-band region accesses. It performs:
 - bit field extract for bit-band loads
 - atomic read-modify-write for bit-band stores.
- Write buffering. The bus matrix contains a one-entry write buffer to decouple bus stalls from the processor core.

Chapter 12 *Bus Interface* describes the bus interfaces.

1.2.4 FPB

You can configure the implementation to include an FPB. The FPB implements hardware breakpoints, and patches accesses from code space to system space. If present, you can configure the FPB to:

- contain six instruction comparators for instruction and literal matching in addition to flash patching. These comparators either remap instruction fetches from code space to system space, or perform a hardware breakpoint.
- contain two comparators that can be used for breakpoints only. These comparators can remap literal accesses from code space to system space.

Chapter 11 *System Debug* describes the FPB.

1.2.5 DWT

You can configure the implementation to include a DWT. If present, you can configure the DWT to incorporate the following debug functionality:

- four comparators that you can configure either as a hardware watchpoint, an ETM trigger, a PC sampler event trigger, or a data address sampler event trigger
- several counters or a data match event trigger for performance profiling
- configurable to emit PC samples at defined intervals, and to emit interrupt event information.

Chapter 11 *System Debug* describes the DWT.

1.2.6 ITM

You can configure the implementation to contain an ITM. The ITM is an application driven trace source that supports application event trace and printf style debugging.

The ITM provides the following sources of trace information:

- Software trace. Software can write directly to ITM stimulus registers. This causes packets to be emitted.
- Hardware trace. These packets are generated by the DWT, and emitted by the ITM.
- Time stamping. Timestamps are emitted relative to packets.

Chapter 11 *System Debug* describes the ITM.

1.2.7 MPU

You can configure the implementation to include an MPU to provide memory protection. The MPU checks access permissions and memory attributes. It contains eight regions, and an optional background region that implements the default memory map attributes.

Chapter 9 *Memory Protection Unit* describes the MPU.

1.2.8 ETM

You can configure the system at implementation to include an ETM. This is a low-cost trace macrocell that supports instruction trace only.

Chapter 14 *Embedded Trace Macrocell* describes the ETM.

1.2.9 AHB-AP

You can configure the implementation to include an AHB-AP.

AHB-AP on page 11-39 describes the AHB-AP.

1.2.10 AHB Trace Macrocell interface

You can configure the system at implementation to include an *AHB Trace Macrocell* (HTM) interface. If you do not enable this option at the time of implementation, the HTM interface does not function because the required logic is not included.

1.2.11 TPIU

You can configure the system at implementation to include an TPIU. The TPIU acts as a bridge between the Cortex-M3 trace data from the ITM, an ETM if present, and an off-chip Trace Port Analyzer.

The implementation options for the TPIU are:

- If the ETM is present in your system, both of the input ports to the TPIU are present. If the ETM is not present but the ITM is, then only one port is used, saving the gate cost of one input FIFO.
- You can replace the ARM TPIU block with a partner-specific CoreSight™ compliant TPIU.
- In a production device, the TPIU might have been removed.

———— **Note** ————

There is no Cortex-M3 trace capability if the TPIU is removed.

—————

Chapter 17 *Trace Port Interface Unit* describes the TPIU.

1.2.12 WIC

You can configure the implementation to include a *Wake-up Interrupt Controller* (WIC).

System power management on page 7-3 describes the WIC functionality.

1.2.13 SW/SWJ-DP

You can configure the processor to have SW-DP or SWJ-DP debug port interfaces. The debug port provides debug access to all registers and memory in the system, including the processor registers.

The implementation options for the SW/SWJ-DP are:

- Your implementation might contain either SW-DP or SWJ-DP.
- You can replace the ARM SW-DP with a partner-specific CoreSight compliant SW-DP.
- You can replace the ARM SWJ-DP with a partner-specific CoreSight compliant SWJ-DP.
- You can include a partner-specific test interface in parallel with SW-DP or SWJ-DP.

———— **Note** ————

The SW/SWJ-DP might not be present in the production device if no debug functionality is present in the implementation.

—————

Chapter 13 *Debug Port* describes the SW/SWJ-DP.

1.2.14 Interrupts

You can configure the number of external interrupts at implementation from 1 to 240. You can configure the number of bits of interrupt priority at implementation from three to eight bits.

1.2.15 Observation

You can configure the system at implementation time to enable the observation of some internal signals. These include the register bank ports and the instruction in the execute stage of the pipeline.

1.2.16 ROM table

The ROM table is modified from that described in *ROM memory table* on page 4-7 if:

- additional debug components have been added into the system
- all debug functionality has been removed from the implementation.

1.3 Execution pipeline stages

The following stages make up the pipeline:

- the Fetch stage
- the Decode stage
- the Execute stage.

Figure 1-2 shows the pipeline stages of the processor, and the pipeline operations that take place at each stage.

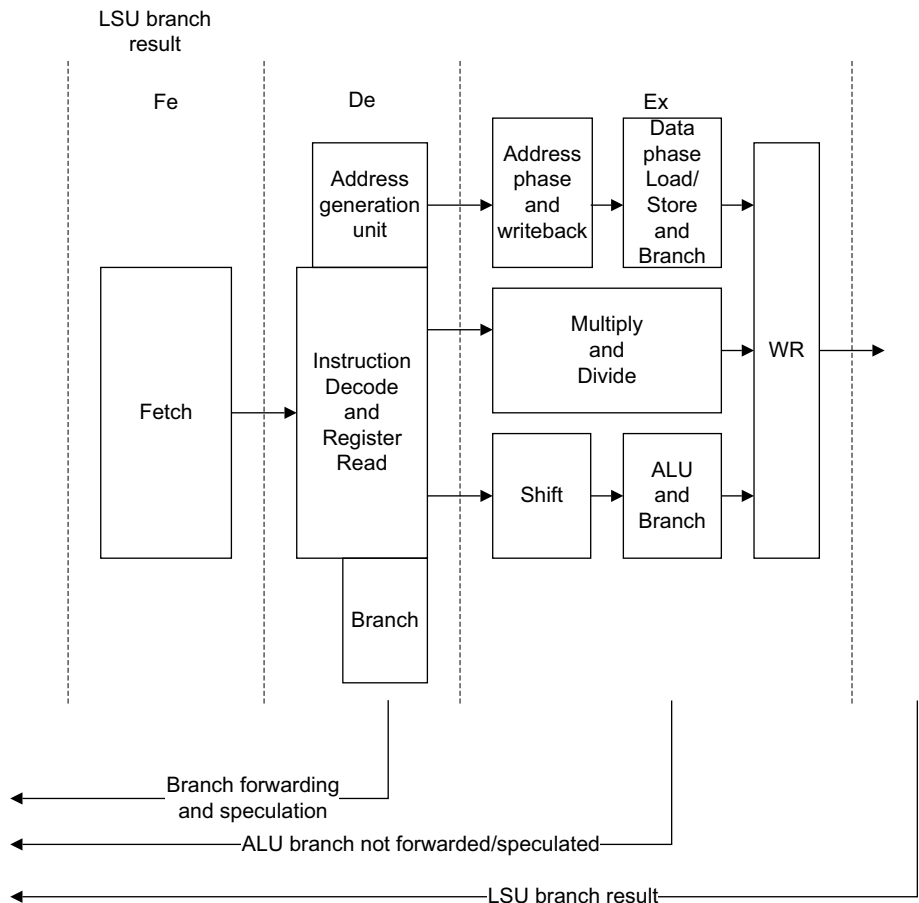


Figure 1-2 Cortex-M3 pipeline stages

The names of the pipeline stages and their functions are:

Fe	Instruction fetch where data is returned from the instruction memory.
De	Instruction decode, generation of LSU address using forwarded register ports, and immediate offset or LR register branch forwarding.
Ex	Instruction execute, single pipeline with multi-cycle stalls, LSU address/data pipelining to AHB interface, multiply/divide, and ALU with branch result.

The pipeline structure provides a pipelined 2-cycle memory access with no ALU usage penalty, address generation forwarding for pointer indirection.

1.4 Prefetch Unit

The purpose of the *Prefetch Unit* (PFU) is to:

- Fetch instructions in advance and forward PC relative branch instructions. Fetches are speculative in the case of conditional branches
- Detect Thumb 32-bit instructions and present these as a single instruction word.
- Perform vector loads.

The PFU fetches instructions from the memory system that can supply one word each cycle. The PFU buffers up to three word fetches in its FIFO, which means that it can buffer up to three Thumb 32-bit instructions or six Thumb instructions.

The majority of branches that are generated as the ALU addition of PC plus immediate are generated no later than the decode phase of the branch opcode. In the case of conditionally executed branches, the address is speculatively presented (consuming a fetch slot on the bus), and the forwarded result determines if the branch path flushes the fetch queue or is preserved.

Short subroutine returns are optimized to take advantage of the forwarding behavior in the case of BX LR.

1.5 Branch target forwarding

The processor forwards certain branch types, by which the memory transaction of the branch is presented at least a cycle earlier than when the opcode reaches execute. Branch forwarding increases the performance of the core, because branches are a significant part of embedded controller applications. Branches affected are PC relative with immediate offset, or use LR as the target register. For conditional branches, by opcode definition or within IT block, that are forwarded, the address must be presented speculatively because the condition evaluation is an internal critical path.

Branch forwarding loses a fetch opportunity if speculated on a conditional opcode, but is mitigated by a three-entry fetch queue and a mix of 16/32-bit opcodes and single cycle ALU. The additional penalty is a cycle of pipeline stalling. The worst case is three 32-bit load/store single opcodes, the instructions word-unaligned, with no data waitstates. The **BRCHSTAT** interface provides information on forwarded branches to conditional execution, the direction if conditional, and a trailing registered evaluation of success of the preceding conditional opcode. For more information on **BRCHSTAT** see *Branch status interface* on page 15-6.

The performance of the core with ICODE registered with prefetch is effectively the same as the core without the branch forwarding interface, around 10% slower. Branch forwarding can be thought of as the internal address generation logic pre-registration to the address interface, increasing flexibility to the memory controller if you have the timing budget to make use of the information a cycle sooner. For example lower MHz power sensitive targets, in 0.13u down to 65nm. Otherwise, you have the flexibility of having access to this early address in your memory controller for lookups before registration to the system.

Branch speculation is more costly against a wait-stated memory because of mispredictions. To avoid this overhead, a rule in the controller that conditional branches are not speculated but instead registered gives subroutine calls and returns the benefits of branch forwarding without the mispredictions penalty. A refinement is to only predict backward conditional branches to accelerate loops. Alternatively, with ARM compilers favouring loops with unconditional branch backwards at the bottom and then conditional branch forward tests on the loop limit, the core fetch queue being ahead at the start of the loop yields good behavior.

The **BRCHSTAT** also includes other information about the next opcode to reach execute. Unlike the forwarded branches where **BRCHSTAT** is incident with the transaction, **BRCHSTAT** with respect to execute opcodes is a hint unrelated to any transaction and can be asserted for multiple cycles. The controller can use this information to suppress additional prefetching because it knows a branch is taken shortly. This helps to avoid any trailing waitstates of the controller prefetch from impacting the branch target when it is generated in execute.

The following scenarios show how you can use branch forwarding and the **BRCHSTAT** control to get the best performance from your memory system. The scenarios focus on the ideal Harvard setup, where instructions execute from ICODE, literals execute from DCODE (unified to ICODE), and stack/heap/application data executes from SYSTEM.

- *Zero waitstate*
- *Zero waitstate, registered fetch interface (ICODE)*
- *One wait state flash*
- *One wait state flash, registered fetch interface (ICODE)*
- *Two wait states flash* on page 1-17.

1.5.1 Zero waitstate

Branch prediction provides approximately 10% gain over not having the feature, and except for extreme cases, the processor has all the benefits of 100% branch prediction but with no penalty from branch speculation.

1.5.2 Zero waitstate, registered fetch interface (ICODE)

Branch forwarding results in more aggressive timing on the ICODE interface. If this bus is a critical path in the system, the ICODE interface might be registered. To avoid an approximate 25% penalty of adding a wait state, you can add a circuit that acts as a single-entry prefetcher.

1.5.3 One wait state flash

Adding wait states to the flash impacts performance of any core. You can use a cache to lessen this penalty, but this has a dramatic effect on determinism and silicon area. A line prefetcher with two line entries can provide comparable performance to a cache using many less gates. 128-bits is a common prefetch width for ARM7 targets because of the 32-bit instruction set. The processor has the benefit of Thumb 32-bit instructions, a mixed 16/32-bit instruction set. This means that a 64-bit prefetch width provides comparable benefits to a 128-bit interface.

1.5.4 One wait state flash, registered fetch interface (ICODE)

If the ICODE interface must be registered, you can reduce the cost of mispredictions to only the slave side of the prefetch controller. The core still loses the opportunity of the fetch queue request on the ICODE interface, as in the zero wait state case. However, the trailing registered **BRCHSTAT[3]** status of the conditional execution can mask the external mispredict on the output of the controller's registered system interface, appearing as an idle cycle.

1.5.5 Two wait states flash

This is the same as one waitstate cases, but with more penalties for branches. The extent to which the compiler tools reduce the overhead of branches, conditioning loops towards the strengths of the hardware, the less the effects of the mismatch between core and memory system speeds. A 128-bit interface is better at this point.

1.6 Store buffers

The processor contains two store buffers:

- Cortex-M3 core LSU store buffer for immediate offset opcode.
- Bus-matrix store buffer for wait states and unaligned transactions.

The core store buffer optimizes the case of `STR rx,[ry,#imm]`, which is common in compiled code. This means that the next opcode can overlap the store's data phase, reducing the opcode to a single cycle from the perspective of the pipeline.

The bus-matrix interconnect within the processor manages the unaligned behavior of the core and bit-banding. The bus-matrix store buffer is useful for resolving system wait-states and unaligned accesses that are split over multiple transactions.

Only transactions marked as bufferable use the store buffers. Stacking operations are inherently non-bufferable and therefore also do not use either of the buffers.

1.7 Product revisions

This section summarizes the differences in functionality between the different releases of this processor:

- *Differences in functionality between r0p0 and r1p0*
- *Differences in functionality between r1p0 and r1p1 on page 1-20*
- *Differences in functionality between r1p1 and r2p0 on page 1-20.*

1.7.1 Differences in functionality between r0p0 and r1p0

In summary, the differences in functionality include:

- Addition of configurable data value comparison to the DWT module. See *DWT* on page 11-13.
- Addition of a MATCHED bit to **DWT_FUNCTION**. See *DWT* on page 11-13.
- Addition of **ETMFIFOFULL** as an input to Cortex-M3. See *ETM interface* on page A-14.
- Addition of **ETMISTALL** as an output to Cortex-M3. See *ETM interface* on page A-14.
- Addition of SWVMode to the ITM. To support SWVMode, **TPIUACTV** and **TPIUBAUD** have been added as outputs from the TPIU and are inputs to the processor. See *ITM* on page 11-30.
- CPUID Base Register VARIANT field changed to indicate Rev1. See *NVIC register descriptions* on page 8-7.
- Cortex-M3 Rev0 Bit-band accesses in BE8 mode required access sizes to be byte. Cortex-M3 Rev1 has been changed so that BE8 bit-band accesses function with any access size.
- Addition of a configuration bit called STKALIGN to ensure that all exceptions have eight-byte stack alignment. See *NVIC register descriptions* on page 8-7.
- Addition of the Auxiliary Fault Status Register at address 0xE00ED3C. To set this register, a 32-bit input bus called **AUXFAULT** has been added. See *NVIC register descriptions* on page 8-7.
- Addition of HTM support. See Chapter 16 *AHB Trace Macrocell Interface*.
- ICode and DCode cacheable and bufferable HPROT values permanently tied to write-through. See *ICode bus interface* on page 12-4 and *DCode bus interface* on page 12-6.

- Addition of a new input called **IFLUSH**. See *Miscellaneous* on page A-4.
- Addition of HMASTER ports. See *DCode interface* on page A-9 and *System bus interface* on page A-10.
- Addition of the SWJ-DP. This is the standard CoreSight™ debug port that combines JTAG-DP and SW-DP. See *About the DP* on page 13-2.
- Addition of DWT_PCSR Register at address 0xE000101C. See *DWT* on page 11-13.
- Addition of a new input called **DNOTITRANS**. See *Unifying the code buses* on page 12-9.
- Errata fixes to the r0p0 release.

1.7.2 Differences in functionality between r1p0 and r1p1

In summary, the differences in functionality include:

- Data value matching for watchpoint generation has been made implementation time configurable. See *DWT* on page 11-13.
- A define has been added to optionally implement architectural clock gating in the ETM. For previous releases the architectural clock gate in the ETM was always present.
- **DAPCLKEN** was required to be a static signal in r0p0 and r1p0. This requirement has been removed for r1p1.
- **SLEEPING** signal now suppressed until current outstanding instruction fetch has completed.
- Errata fixes to the r1p0 release.

1.7.3 Differences in functionality between r1p1 and r2p0

In summary, the differences in functionality include:

- Implementation time options have been added to select between different levels of debug and trace support. This has replaced the previous TIEOFF_FPEN and TIEOFF_TRCENA options.
- New implementation option to enable the resetting of all registers within the processor.
- Architectural clock gating inclusion is now controlled using one implementation option.

- **DBGRESTART** input and **DBGRESTARTED** output has been added for use in debugging multi-core systems. See the *ARMv7-M Architecture Reference Manual* for more information.
- **SLEEPHOLDREQn** input and **SLEEPHOLDACKn** have been added to enable the extension of SLEEPING. See *Extending sleep* on page 7-5.
- The APB interface has been upgraded from v2.0 to v3.0. See *External private peripheral interface* on page 12-10.
- A new output signal called **INTERNALSTATE** has been added that enables observation of some of the internal state of the core if the **OBSERVATION** implementation option is used.
- An Auxiliary Control Register has been added with new functionality disable bits to:
 - stop interruption of load/store multiples, divides and multiplies
 - stop IT folding
 - disable the write buffers in Cortex-M3 for default memory map accesses.
 For details on the Auxiliary Control Register see *Auxiliary Control Register* on page 8-8.
- The **STKALIGN** bit reset value in the Configuration and Control Register at address 0xE000ED14 has been inverted. The reset value is now 1, which means that the stack frame is 8-byte aligned by default. See *Configuration Control Register* on page 8-26.
- Addition of a Wake-up Interrupt Controller to minimize logic in always clocked domain during sleep. For details see *Using the Wake-up Interrupt Controller* on page 7-6.
- Addition of **FIXHMASTERTYPE** pin to prevent debugger marking AHB transactions as core data side if required.
- Errata fixes to the r1p1 release.

Chapter 2

Programmer's Model

This chapter describes the processor programmer's model. It contains the following sections:

- *About the programmer's model* on page 2-2
- *Privileged access and user access* on page 2-3
- *Registers* on page 2-4
- *Data types* on page 2-10
- *Memory formats* on page 2-11
- *Instruction set summary* on page 2-13.

2.1 About the programmer's model

The processor implements the ARMv7-M architecture. This includes all the 16-bit Thumb instructions and the base 32-bit Thumb instructions. The processor cannot execute ARM instructions. For more information about the ARMv7-M Thumb instruction set see the *ARMv7-M Architecture Reference Manual*.

2.1.1 Operating modes

The processor supports two modes of operation, Thread mode and Handler mode:

- Thread mode is entered on Reset, and can be entered as a result of an exception return. Privileged and User (Unprivileged) code can run in Thread mode.
- Handler mode is entered as a result of an exception. All code is privileged in Handler mode.

2.1.2 Operating states

The processor can operate in one of two operating states:

- Thumb state. This is normal execution running 16-bit and 32-bit halfword aligned Thumb instructions.
- Debug State. This is the state when in halting debug.

2.2 Privileged access and user access

Code can execute as privileged or unprivileged. Unprivileged execution limits or excludes access to some resources. Privileged execution has access to all resources. Handler mode is always privileged. Thread mode can be privileged or unprivileged.

Thread mode is privileged out of reset, but you can change it to user or unprivileged by setting the CONTROL[0] bit using the MSR instruction. User access prevents:

- use of some instructions such as CPS to set FAULTMASK and PRIMASK
- access to most registers in *System Control Space* (SCS).

When Thread mode has been changed from privileged to user, it cannot change itself back to privileged. Only a Handler can change the privilege of Thread mode. Handler mode is always privileged.

2.2.1 Main stack and process stack

Out of reset, all code uses the main stack. An exception handler such as SVC can change the stack used by Thread mode from main stack to process stack by changing the EXC_RETURN value it uses on exit. All exceptions continue to use the main stack. The stack pointer, r13, is a banked register that switches between SP_main and SP_process. Only one stack, the process stack or the main stack, is visible, using r13, at any time.

It is also possible to switch from main stack to process stack while in Thread mode by writing to CONTROL[1] using the MSR instruction, in addition to being selectable using the EXC_RETURN value from an exit from Handler mode.

2.3 Registers

The processor has the following 32-bit registers:

- 13 general-purpose registers, r0-r12
- stack point alias of banked registers, SP_process and SP_main
- link register, r14
- program counter, r15
- one program status register, xPSR.

Figure 2-1 shows the processor register set.

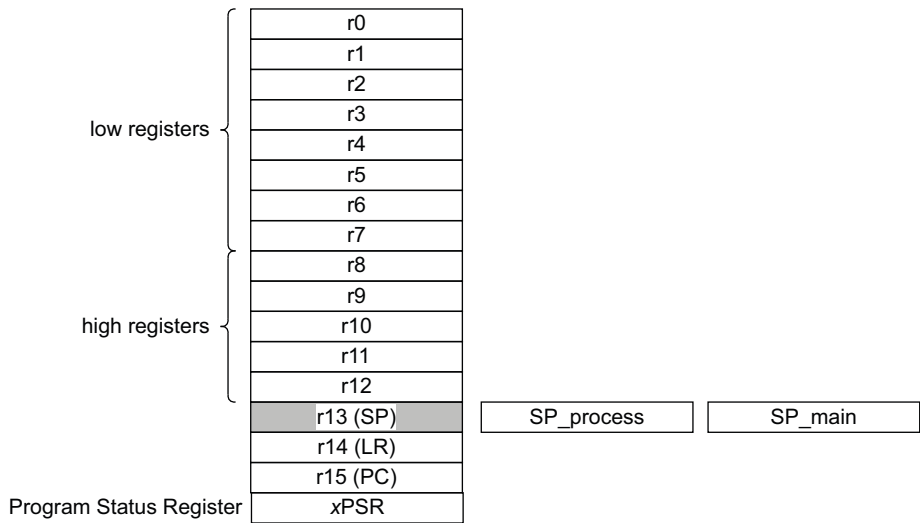


Figure 2-1 Processor register set

2.3.1 General-purpose registers

The general-purpose registers r0-r12 have no special architecturally-defined uses. Most instructions that can specify a general-purpose register can specify r0-r12.

- Low registers** Registers r0-r7 are accessible by all instructions that specify a general-purpose register.
- High registers** Registers r8-r12 are accessible by all 32-bit instructions that specify a general-purpose register.
- Registers r8-r12 are not accessible by all 16-bit instructions.

The r13, r14, and r15 registers have the following special functions:

Stack pointer	<p>Register r13 is used as the <i>Stack Pointer</i> (SP). Because the SP ignores writes to bits [1:0], it is autoaligned to a word, four-byte boundary.</p> <p>Handler mode always uses SP_main, but you can configure Thread mode to use either SP_main or SP_process.</p>
Link register	<p>Register r14 is the subroutine <i>Link Register</i> (LR).</p> <p>The LR receives the return address from PC when a <i>Branch and Link</i> (BL) or <i>Branch and Link with Exchange</i> (BLX) instruction is executed.</p> <p>The LR is also used for exception return.</p> <p>At all other times, you can treat r14 as a general-purpose register.</p>
Program counter	<p>Register r15 is the <i>Program Counter</i> (PC).</p> <p>Bit [0] is always 0, so instructions are always aligned to word or halfword boundaries.</p>

2.3.2 Special-purpose Program Status Registers (xPSR)

Processor status at the system level breaks down into three categories:

- *Application PSR*
- *Interrupt PSR* on page 2-6
- *Execution PSR* on page 2-7.

They can be accessed as individual registers, a combination of any two from three, or a combination of all three using the *Move to Register from Status* (MRS) and MSR instructions.

Application PSR

The *Application PSR* (APSR) contains the condition code flags. Before entering an exception, the processor saves the condition code flags on the stack. You can access the APSR with the MSR(2) and MRS(2) instructions.

Figure 2-2 on page 2-6 shows the bit assignments of the APSR.

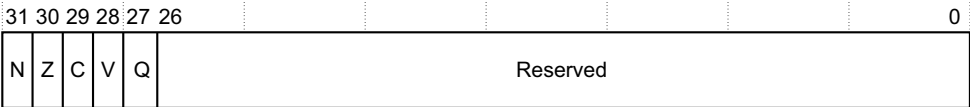


Figure 2-2 Application Program Status Register bit assignments

Table 2-1 describes the bit assignments of the APSR.

Table 2-1 Application Program Status Register bit assignments

Field	Name	Definition
[31]	N	Negative or less than flag: 1 = result negative or less than 0 = result positive or greater than.
[30]	Z	Zero flag: 1 = result of 0 0 = nonzero result.
[29]	C	Carry/borrow flag: 1 = carry or borrow 0 = no carry or borrow.
[28]	V	Overflow flag: 1 = overflow 0 = no overflow.
[27]	Q	Sticky saturation flag.
[26:0]	-	Reserved.

Interrupt PSR

The *Interrupt PSR* (IPSR) contains the *Interrupt Service Routine* (ISR) number of the current exception activation.

Figure 2-2 shows the bit assignments of the IPSR.

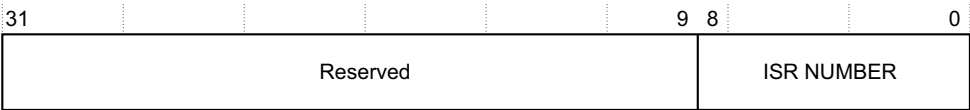


Figure 2-3 Interrupt Program Status Register bit assignments

Table 2-2 describes the bit assignments of the IPSR.

Table 2-2 Interrupt Program Status Register bit assignments

Field	Name	Definition
[31:9]	-	Reserved.
[8:0]	ISR NUMBER	Number of pre-empted exception. Base level = 0 NMI = 2 SVCall = 11 INTISR[0] = 16 INTISR[1] = 17 . . . INTISR[15] = 31 . . . INTISR[239] = 255

Execution PSR

The *Execution PSR* (EPSR) contains two overlapping fields:

- the *Interruptible-Continuable Instruction* (ICI) field for interrupted load multiple and store multiple instructions
- the execution state field for the *If-Then* (IT) instruction, and the *Thumb state bit* (T-bit).

Interruptible-continuable instruction field

Load Multiple (LDM) operations and *Store Multiple* (STM) operations are interruptible. The ICI field of the EPSR holds the information required to continue the load or store multiple from the point that the interrupt occurred.

If-then state field

The IT field of the EPSR contain the execution state bits for the If-Then instruction.

Note

Because the ICI field and the IT field overlap, load or store multiples within an If-Then block cannot be interrupt-continued.

Figure 2-4 shows the bit assignments of the EPSR.

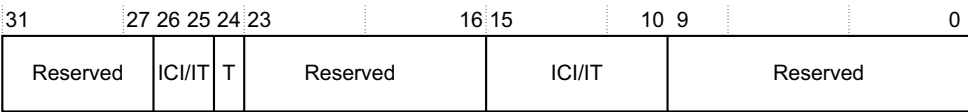


Figure 2-4 Execution Program Status Register

The EPSR is not directly accessible. Two events can modify the EPSR:

- an interrupt occurring during an LDM or STM instruction
- execution of the If-Then instruction.

Table 2-3 describes the bit assignments of the EPSR.

Table 2-3 Bit functions of the EPSR

Field	Name	Definition
[31:27]	-	Reserved.
[26:25], [15:10]	ICI	Interruptible-continuable instruction bits. When an interrupt occurs during an LDM or STM operation, the multiple operation stops temporarily. The EPSR uses bits [15:12] to store the number of the next register operand in the multiple operation. After servicing the interrupt, the processor returns to the register pointed to by [15:12] and resumes the multiple operation.
[26:25], [15:10]	IT	If-Then bits. These are the execution state bits of the If-Then instruction. They contain the number of instructions in the if-then block and the conditions for their execution.
[24]	T	The T-bit can be cleared using an interworking instruction where bit [0] of the written PC is 0. It can also be cleared by unstacking from an exception where the stacked T bit is 0. Executing an instruction while the T bit is clear causes an INVSTATE exception.
[23:16]	-	Reserved.
[9:0]	-	Reserved.

Base register update in LDM and STM operations

There are cases when an LDM or STM updates the base register:

- When the instruction specifies base register write-back, the base register changes to the updated address. An abort restores the original base value.
- When the base register is in the register list of an LDM, and is not the last register in the list, the base register changes to the loaded value.

An LDM/STM is restarted rather than continued if:

- the LDM/STM faults
- the LDM/STM is inside an IT.

If an LDM has completed a base load, it is continued from before the base load.

Saved xPSR bits

On entering an exception, the processor saves the combined information from the three status registers on the stack. The stacked xPSR also contains information about whether the stack was 8-byte aligned or not depending on the value of STKALIGN in the Configuration Control Register. This information is stored in bit [9] of the xPSR on the stack, and it is a 1 if the stack was forced to be 8-byte aligned.

2.4 Data types

The processor supports the following data types:

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

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.

2.5 Memory formats

The 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 access data words in memory in little-endian format or big-endian format. It always accesses code in little-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. 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. The byte at address 0 of the memory system connects to data lines 31-24.

Figure 2-5 on page 2-12 shows the difference between little-endian and big-endian memory formats.

The processor contains a configuration pin, **BIGEND**, that enables you to select either the little-endian or BE-8 big-endian format. This configuration pin is sampled on reset. You cannot change endianness when out of reset.

Note

- Accesses to *System Control Space* (SCS) are always little endian.
 - Attempts to change endianness while not in reset are ignored.
 - *Private Peripheral Bus* (PPB) space is little-endian, irrespective of the setting of **BIGEND**.
-

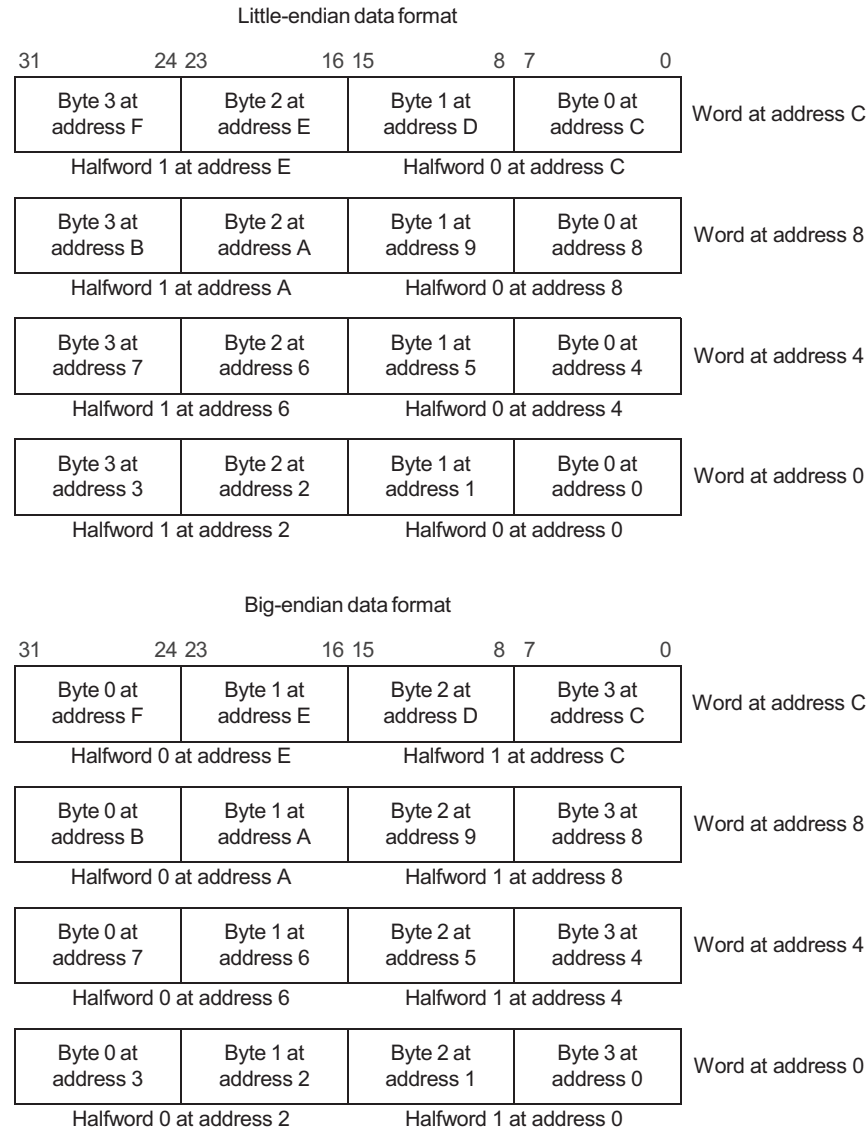


Figure 2-5 Little-endian and big-endian memory formats

2.6 Instruction set summary

This section provides:

- a summary of the processor 16-bit instructions
- a summary of the processor 32-bit instructions.

Table 2-4 lists the 16-bit Cortex-M3 instructions.

Table 2-4 16-bit Cortex-M3 instruction summary

Operation	Assembler
Add register value and C flag to register value	ADC <Rd>, <Rm>
Add immediate 3-bit value to register	ADD <Rd>, <Rn>, #<immed_3>
Add immediate 8-bit value to register	ADD <Rd>, #<immed_8>
Add low register value to low register value	ADD <Rd>, <Rn>, <Rm>
Add high register value to low or high register value	ADD <Rd>, <Rm>
Add 4* (immediate 8-bit value) with PC to register	ADD <Rd>, PC, #<immed_8> * 4
Add 4* (immediate 8-bit value) with SP to register	ADD <Rd>, SP, #<immed_8> * 4
Add 4* (immediate 7-bit value) to SP	ADD SP, #<immed_7> * 4
Bitwise AND register values	AND <Rd>, <Rm>
Arithmetic shift right by immediate number	ASR <Rd>, <Rm>, #<immed_5>
Arithmetic shift right by number in register	ASR <Rd>, <Rs>
Branch conditional	B<cond> <target address>
Branch unconditional	B <target_address>
Bit clear	BIC <Rd>, <Rm>
Software breakpoint	BKPT <immed_8>
Branch with link	BL <Rm>
Branch with link and exchange	BLX <Rm>
Branch and exchange	BX <Rm>
Compare not zero and branch	CBNZ <Rn>, <label>
Compare zero and branch	CBZ <Rn>, <label>

Table 2-4 16-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Compare negation of register value with another register value	CMN <Rn>, <Rm>
Compare immediate 8-bit value	CMP <Rn>, #<immed_8>
Compare registers	CMP <Rn>, <Rm>
Compare high register to low or high register	CMP <Rn>, <Rm>
Change processor state	CPS <effect>, <iflags>
Copy high or low register value to another high or low register	CPY <Rd> <Rm>
Bitwise exclusive OR register values	EOR <Rd>, <Rm>
Condition the following instruction	IT <cond>
Condition the following two instructions	IT<x> <cond>
Condition the following three instructions	IT<x><y> <cond>
Condition the following four instructions	IT<x><y><z> <cond>
Multiple sequential memory word loads	LDMIA <Rn>!, <registers>
Load memory word from base register address + 5-bit immediate offset	LDR <Rd>, [<Rn>, #<immed_5> * 4]
Load memory word from base register address + register offset	LDR <Rd>, [<Rn>, <Rm>]
Load memory word from PC address + 8-bit immediate offset	LDR <Rd>, [PC, #<immed_8> * 4]
Load memory word from SP address + 8-bit immediate offset	LDR, <Rd>, [SP, #<immed_8> * 4]
Load memory byte [7:0] from register address + 5-bit immediate offset	LDRB <Rd>, [<Rn>, #<immed_5>]
Load memory byte [7:0] from register address + register offset	LDRB <Rd>, [<Rn>, <Rm>]
Load memory halfword [15:0] from register address + 5-bit immediate offset	LDRH <Rd>, [<Rn>, #<immed_5> * 2]
Load halfword [15:0] from register address + register offset	LDRH <Rd>, [<Rn>, <Rm>]
Load signed byte [7:0] from register address + register offset	LDRSB <Rd>, [<Rn>, <Rm>]
Load signed halfword [15:0] from register address + register offset	LDRSH <Rd>, [<Rn>, <Rm>]
Logical shift left by immediate number	LSL <Rd>, <Rm>, #<immed_5>
Logical shift left by number in register	LSL <Rd>, <Rs>
Logical shift right by immediate number	LSR <Rd>, <Rm>, #<immed_5>
Logical shift right by number in register	LSR <Rd>, <Rs>

Table 2-4 16-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Move immediate 8-bit value to register	MOV <Rd>, #<immed_8>
Move low register value to low register	MOV <Rd>, <Rn>
Move high or low register value to high or low register	MOV <Rd>, <Rm>
Multiply register values	MUL <Rd>, <Rm>
Move complement of register value to register	MVN <Rd>, <Rm>
Negate register value and store in register	NEG <Rd>, <Rm>
No operation	NOP <c>
Bitwise logical OR register values	ORR <Rd>, <Rm>
Pop registers from stack	POP <registers>
Pop registers and PC from stack	POP <registers, PC>
Push registers onto stack	PUSH <registers>
Push LR and registers onto stack	PUSH <registers, LR>
Reverse bytes in word and copy to register	REV <Rd>, <Rn>
Reverse bytes in two halfwords and copy to register	REV16 <Rd>, <Rn>
Reverse bytes in low halfword [15:0], sign-extend, and copy to register	REVSH <Rd>, <Rn>
Rotate right by amount in register	ROR <Rd>, <Rs>
Subtract register value and C flag from register value	SBC <Rd>, <Rm>
Send event	SEV <c>
Store multiple register words to sequential memory locations	STMIA <Rn>!, <registers>
Store register word to register address + 5-bit immediate offset	STR <Rd>, [<Rn>, #<immed_5> * 4]
Store register word to register address	STR <Rd>, [<Rn>, <Rm>]
Store register word to SP address + 8-bit immediate offset	STR <Rd>, [SP, #<immed_8> * 4]
Store register byte [7:0] to register address + 5-bit immediate offset	STRB <Rd>, [<Rn>, #<immed_5>]
Store register byte [7:0] to register address	STRB <Rd>, [<Rn>, <Rm>]
Store register halfword [15:0] to register address + 5-bit immediate offset	STRH <Rd>, [<Rn>, #<immed_5> * 2]

Table 2-4 16-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Store register halfword [15:0] to register address + register offset	STRH <Rd>, [<Rn>, <Rm>]
Subtract immediate 3-bit value from register	SUB <Rd>, <Rn>, #<immed_3>
Subtract immediate 8-bit value from register value	SUB <Rd>, #<immed_8>
Subtract register values	SUB <Rd>, <Rn>, <Rm>
Subtract 4 (immediate 7-bit value) from SP	SUB SP, #<immed_7> * 4
Operating system service call with 8-bit immediate call code	SVC <immed_8>
Extract byte [7:0] from register, move to register, and sign-extend to 32 bits	SXTB <Rd>, <Rm>
Extract halfword [15:0] from register, move to register, and sign-extend to 32 bits	SXTH <Rd>, <Rm>
Test register value for set bits by ANDing it with another register value	TST <Rn>, <Rm>
Extract byte [7:0] from register, move to register, and zero-extend to 32 bits	UXTB <Rd>, <Rm>
Extract halfword [15:0] from register, move to register, and zero-extend to 32 bits	UXTH <Rd>, <Rm>
Wait for event	WFE <C>
Wait for interrupt	WFI <C>

Table 2-5 lists the 32-bit Cortex-M3 instructions.

Table 2-5 32-bit Cortex-M3 instruction summary

Operation	Assembler
Add register value, immediate 12-bit value, and C bit	ADC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12>
Add register value, shifted register value, and C bit	ADC{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Add register value and immediate 12-bit value	ADD{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Add register value and shifted register value	ADD{S}.W <Rd>, <Rm>{, <shift>}
Add register value and immediate 12-bit value	ADDW.W <Rd>, <Rn>, #<immed_12>
Bitwise AND register value with immediate 12-bit value	AND{S}.W <Rd>, <Rn>, #<modify_constant(immed_12>
Bitwise AND register value with shifted register value	AND{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Arithmetic shift right by number in register	ASR{S}.W <Rd>, <Rn>, <Rm>

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Conditional branch	B{cond}.W <label>
Clear bit field	BFC.W <Rd>, #<lsb>, #<width>
Insert bit field from one register value into another	BFI.W <Rd>, <Rn>, #<lsb>, #<width>
Bitwise AND register value with complement of immediate 12-bit value	BIC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Bitwise AND register value with complement of shifted register value	BIC{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Branch with link	BL <label>
Branch with link (immediate)	BL<c> <label>
Unconditional branch	B.W <label>
Clear exclusive clears the local record of the executing processor that an address has had a request for an exclusive access.	CLREX <c>
Return number of leading zeros in register value	CLZ.W <Rd>, <Rn>
Compare register value with two's complement of immediate 12-bit value	CMN.W <Rn>, #<modify_constant(immed_12)>
Compare register value with two's complement of shifted register value	CMN.W <Rn>, <Rm>{, <shift>}
Compare register value with immediate 12-bit value	CMP.W <Rn>, #<modify_constant(immed_12)>
Compare register value with shifted register value	CMP.W <Rn>, <Rm>{, <shift>}
Data memory barrier	DMB <c>
Data synchronization barrier	DSB <c>
Exclusive OR register value with immediate 12-bit value	EOR{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Exclusive OR register value with shifted register value	EOR{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Instruction synchronization barrier	ISB <c>
Load multiple memory registers, increment after or decrement before	LDM{IA DB}.W <Rn>{!}, <registers>

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Memory word from base register address + immediate 12-bit offset	LDR.W <Rxf>, [<Rn>, #<offset_12>]
Memory word to PC from register address + immediate 12-bit offset	LDR.W PC, [<Rn>, #<offset_12>]
Memory word to PC from base register address immediate 8-bit offset, postindexed	LDR.W PC, [Rn], #<+/-<offset_8>
Memory word from base register address immediate 8-bit offset, postindexed	LDR.W <Rxf>, [<Rn>], #+/-<offset_8>
Memory word from base register address immediate 8-bit offset, preindexed	LDR.W <Rxf>, [<Rn>, #<+/-<offset_8>]! LDRT.W <Rxf>, [<Rn>, #<offset_8>]
Memory word to PC from base register address immediate 8-bit offset, preindexed	LDR.W PC, [<Rn>, #+/-<offset_8>]!
Memory word from register address shifted left by 0, 1, 2, or 3 places	LDR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Memory word to PC from register address shifted left by 0, 1, 2, or 3 places	LDR.W PC, [<Rn>, <Rm>{, LSL #<shift>}]
Memory word from PC address immediate 12-bit offset	LDR.W <Rxf>, [PC, #+/-<offset_12>]
Memory word to PC from PC address immediate 12-bit offset	LDR.W PC, [PC, #+/-<offset_12>]
Memory byte [7:0] from base register address + immediate 12-bit offset	LDRB.W <Rxf>, [<Rn>, #<offset_12>]
Memory byte [7:0] from base register address immediate 8-bit offset, postindexed	LDRB.W <Rxf>. [<Rn>], #+/-<offset_8>
Memory byte [7:0] from register address shifted left by 0, 1, 2, or 3 places	LDRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Memory byte [7:0] from base register address immediate 8-bit offset, preindexed	LDRB.W <Rxf>, [<Rn>, #<+/-<offset_8>]!
Memory byte from PC address immediate 12-bit offset	LDRB.W <Rxf>, [PC, #+/-<offset_12>]
Memory doubleword from register address 8-bit offset 4, preindexed	LDRD.W <Rxf>, <Rxf2>, [<Rn>, #+/-<offset_8> * 4]{}!
Memory doubleword from register address 8-bit offset 4, postindexed	LDRD.W <Rxf>, <Rxf2>, [<Rn>], #+/-<offset_8> * 4

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Load register exclusive calculates an address from a base register value and an immediate offset, loads a word from memory, writes it to a register	LDREX<c> <Rt>, [<Rn>{, #<imm>}]
Load register exclusive halfword calculates an address from a base register value and an immediate offset, loads a halfword from memory, writes it to a register	LDREXH<c> <Rt>, [<Rn>{, #<imm>}]
Load register exclusive byte calculates an address from a base register value and an immediate offset, loads a byte from memory, writes it to a register	LDREXB<c> <Rt>, [<Rn>{, #<imm>}]
Memory halfword [15:0] from base register address + immediate 12-bit offset	LDRH.W <Rxf>, [<Rn>, #<offset_12>]
Memory halfword [15:0] from base register address immediate 8-bit offset, preindexed	LDRH.W <Rxf>, [<Rn>, #<+/-offset_8>]!
Memory halfword [15:0] from base register address immediate 8-bit offset, postindexed	LDRH.W <Rxf>. [<Rn>], #<+/-offset_8>
Memory halfword [15:0] from register address shifted left by 0, 1, 2, or 3 places	LDRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Memory halfword from PC address immediate 12-bit offset	LDRH.W <Rxf>, [PC, #<+/-offset_12>]
Memory signed byte [7:0] from base register address + immediate 12-bit offset	LDRSB.W <Rxf>, [<Rn>, #<offset_12>]
Memory signed byte [7:0] from base register address immediate 8-bit offset, postindexed	LDRSB.W <Rxf>. [<Rn>], #<+/-offset_8>
Memory signed byte [7:0] from base register address immediate 8-bit offset, preindexed	LDRSB.W <Rxf>, [<Rn>, #<+/-offset_8>]!
Memory signed byte [7:0] from register address shifted left by 0, 1, 2, or 3 places	LDRSB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Memory signed byte from PC address immediate 12-bit offset	LDRSB.W <Rxf>, [PC, #<+/-offset_12>]
Memory signed halfword [15:0] from base register address + immediate 12-bit offset	LDRSH.W <Rxf>, [<Rn>, #<offset_12>]
Memory signed halfword [15:0] from base register address immediate 8-bit offset, postindexed	LDRSH.W <Rxf>. [<Rn>], #<+/-offset_8>

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Memory signed halfword [15:0] from base register address immediate 8-bit offset, preindexed	LDRSH.W <Rxf>, [<Rn>, #<+/-<offset_8>]!
Memory signed halfword [15:0] from register address shifted left by 0, 1, 2, or 3 places	LDRSH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Memory signed halfword from PC address immediate 12-bit offset	LDRSH.W <Rxf>, [PC, #<+/-<offset_12>]
Logical shift left register value by number in register	LSL{S}.W <Rd>, <Rn>, <Rm>
Logical shift right register value by number in register	LSR{S}.W <Rd>, <Rn>, <Rm>
Multiply two signed or unsigned register values and add the low 32 bits to a register value	MLA.W <Rd>, <Rn>, <Rm>, <Racc>
Multiply two signed or unsigned register values and subtract the low 32 bits from a register value	MLS.W <Rd>, <Rn>, <Rm>, <Racc>
Move immediate 12-bit value to register	MOV{S}.W <Rd>, #<modify_constant(immed_12)>
Move shifted register value to register	MOV{S}.W <Rd>, <Rm>{, <shift>}
Move immediate 16-bit value to top halfword [31:16] of register	MOVT.W <Rd>, #<immed_16>
Move immediate 16-bit value to bottom halfword [15:0] of register and clear top halfword [31:16]	MOVW.W <Rd>, #<immed_16>
Move to register from status	MRS<c> <Rd>, <psr>
Move to status register	MSR<c> <psr>_<fields>, <Rn>
Multiply two signed or unsigned register values	MUL.W <Rd>, <Rn>, <Rm>
No operation	NOP.W
Logical OR NOT register value with immediate 12-bit value	ORN{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Logical OR NOT register value with shifted register value	ORN{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Logical OR register value with immediate 12-bit value	ORR{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Logical OR register value with shifted register value	ORR{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Reverse bit order	RBIT.W <Rd>, <Rm>
Reverse bytes in word	REV.W <Rd>, <Rm>

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Reverse bytes in each halfword	REV16.W <Rd>, <Rn>
Reverse bytes in bottom halfword and sign-extend	REVSH.W <Rd>, <Rn>
Rotate right by number in register	ROR{S}.W <Rd>, <Rn>, <Rm>
Rotate right with extend	RRX{S}.W <Rd>, <Rm>
Subtract a register value from an immediate 12-bit value	RSB{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Subtract a register value from a shifted register value	RSB{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Subtract immediate 12-bit value and C bit from register value	SBC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Subtract shifted register value and C bit from register value	SBC{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Copy selected bits to register and sign-extend	SBFX.W <Rd>, <Rn>, #<lsb>, #<width>
Signed divide	SDIV<C> <Rd>, <Rn>, <Rm>
Send event	SEV<C>
Multiply signed words and add signed-extended value to 2-register value	SMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm>
Multiply two signed register values	SMULL.W <RdLo>, <RdHi>, <Rn>, <Rm>
Signed saturate	SSAT.W <C> <Rd>, #<imm>, <Rn>{, <shift>}
Multiple register words to consecutive memory locations	STM{IA DB}.W <Rn>{!}, <registers>
Register word to register address + immediate 12-bit offset	STR.W <Rxf>, [<Rn>, #<offset_12>]
Register word to register address immediate 8-bit offset, postindexed	STR.W <Rxf>, [<Rn>], #+/-<offset_8>
Register word to register address shifted by 0, 1, 2, or 3 places	STR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Register word to register address immediate 8-bit offset, preindexed Store, preindexed	STR.W <Rxf>, [<Rn>, #+/-<offset_8>]{} STRT.W <Rxf>, [<Rn>, #<offset_8>]
Register byte [7:0] to register address immediate 8-bit offset, preindexed	STRB{T}.W <Rxf>, [<Rn>, #+/-<offset_8>]{} STRT.W <Rxf>, [<Rn>, #<offset_8>]
Register byte [7:0] to register address + immediate 12-bit offset	STRB.W <Rxf>, [<Rn>, #<offset_12>]

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Register byte [7:0] to register address immediate 8-bit offset, postindexed	STRB.W <Rxf>, [<Rn>], #+/-<offset_8>
Register byte [7:0] to register address shifted by 0, 1, 2, or 3 places	STRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Store doubleword, preindexed	STRD.W <Rxf>, <Rxf2>, [<Rn>, #+/-<offset_8> * 4]{!}
Store doubleword, postindexed	STRD.W <Rxf>, <Rxf2>, [<Rn>, #+/-<offset_8> * 4]
Store register exclusive calculates an address from a base register value and an immediate offset, and stores a word from a register to memory if the executing processor has exclusive access to the memory addressed.	STREX <c> <Rd>, <Rt>, [<Rn>{, #<imm>}]
Store register exclusive byte derives an address from a base register value, and stores a byte from a register to memory if the executing processor has exclusive access to the memory addressed	STREXB <c> <Rd>, <Rt>, [<Rn>]
Store register exclusive halfword derives an address from a base register value, and stores a halfword from a register to memory if the executing processor has exclusive access to the memory addressed.	STREXH <c> <Rd>, <Rt>, [<Rn>]
Register halfword [15:0] to register address + immediate 12-bit offset	STRH.W <Rxf>, [<Rn>, #<offset_12>]
Register halfword [15:0] to register address shifted by 0, 1, 2, or 3 places	STRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
Register halfword [15:0] to register address immediate 8-bit offset, preindexed	STRH{T}.W <Rxf>, [<Rn>, #+/-<offset_8>]{!}
Register halfword [15:0] to register address immediate 8-bit offset, postindexed	STRH.W <Rxf>, [<Rn>], #+/-<offset_8>
Subtract immediate 12-bit value from register value	SUB{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
Subtract shifted register value from register value	SUB{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
Subtract immediate 12-bit value from register value	SUBW.W <Rd>, <Rn>, #<immed_12>
Sign extend byte to 32 bits	SXTB.W <Rd>, <Rm>{, <rotation>}
Sign extend halfword to 32 bits	SXTH.W <Rd>, <Rm>{, <rotation>}

Table 2-5 32-bit Cortex-M3 instruction summary (continued)

Operation	Assembler
Table branch byte	TBB [<Rn>, <Rm>]
Table branch halfword	TBH [<Rn>, <Rm>, LSL #1]
Exclusive OR register value with immediate 12-bit value	TEQ.W <Rn>, #<modify_constant(immed_12)>
Exclusive OR register value with shifted register value	TEQ.W <Rn>, <Rm>{, <shift>}
Logical AND register value with 12-bit immediate value	TST.W <Rn>, #<modify_constant(immed_12)>
Logical AND register value with shifted register value	TST.W <Rn>, <Rm>{, <shift>}
Copy bit field from register value to register and zero-extend to 32 bits	UBFX.W <Rd>, <Rn>, #<lsb>, #<width>
Unsigned divide	UDIV<C> <Rd>, <Rn>, <Rm>
Multiply two unsigned register values and add to a 2-register value	UMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm>
Multiply two unsigned register values	UMULL.W <RdLo>, <RdHi>, <Rn>, <Rm>
Unsigned saturate	USAT <C> <Rd>, #<imm>, <Rn>{, <shift>}
Copy unsigned byte to register and zero-extend to 32 bits	UXTB.W <Rd>, <Rm>{, <rotation>}
Copy unsigned halfword to register and zero-extend to 32 bits	UXTH.W <Rd>, <Rm>{, <rotation>}
Wait for event	WFE.W
Wait for interrupt	WFI.W

Chapter 3

System Control

This chapter describes the registers that program the processor. It contains the following section:

- *Summary of processor registers* on page 3-2.

3.1 Summary of processor registers

This section describes the registers that control functionality. It contains the following:

- *Nested Vectored Interrupt Controller registers*
- *Core debug registers* on page 3-5
- *System debug registers* on page 3-6
- *Debug interface port registers* on page 3-10
- *Memory Protection Unit registers* on page 3-11
- *Trace Port Interface Unit registers* on page 3-12
- *Embedded Trace Macrocell registers* on page 3-13.

3.1.1 Nested Vectored Interrupt Controller registers

Table 3-1 gives a summary of the *Nested Vectored Interrupt Controller* (NVIC) registers. For a detailed description of the NVIC registers, see Chapter 8 *Nested Vectored Interrupt Controller*.

Table 3-1 NVIC registers

Name of register	Type	Address	Reset value
Interrupt Control Type Register	Read-only	0xE000E004	a
Auxiliary Control Register	Read/write	0xE000E008	0x0
SysTick Control and Status Register	Read/write	0xE000E010	0x00000000
SysTick Reload Value Register	Read/write	0xE000E014	Unpredictable
SysTick Current Value Register	Read/write clear	0xE000E018	Unpredictable
SysTick Calibration Value Register	Read-only	0xE000E01C	STCALIB
Irq 0 to 31 Set Enable Register	Read/write	0xE000E100	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 224 to 239 Set Enable Register	Read/write	0xE000E11C	0x00000000
Irq 0 to 31 Clear Enable Register	Read/write	0xE000E180	0x00000000
.	.	.	.
.	.	.	.

Table 3-1 NVIC registers (continued)

Name of register	Type	Address	Reset value
.	.	.	.
Irq 224 to 239 Clear Enable Register	Read/write	0xE000E19C	0x00000000
Irq 0 to 31 Set Pending Register	Read/write	0xE000E200	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 224 to 239 Set Pending Register	Read/write	0xE000E21C	0x00000000
Irq 0 to 31 Clear Pending Register	Read/write	0xE000E280	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 224 to 239 Clear Pending Register	Read/write	0xE000E29C	0x00000000
Irq 0 to 31 Active Bit Register	Read-only	0xE000E300	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 224 to 239 Active Bit Register	Read-only	0xE000E31C	0x00000000
Irq 0 to 3 Priority Register	Read/write	0xE000E400	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 236 to 239 Priority Register	Read/write	0xE000E4EC	0x00000000
CPUID Base Register	Read-only	0xE000ED00	0x412FC230
Interrupt Control State Register	Read/write or read-only	0xE000ED04	0x00000000
Vector Table Offset Register	Read/write	0xE000ED08	0x00000000

Table 3-1 NVIC registers (continued)

Name of register	Type	Address	Reset value
Application Interrupt/Reset Control Register	Read/write	0xE000ED0C	0x00000000
System Control Register	Read/write	0xE000ED10	0x00000000
Configuration Control Register	Read/write	0xE000ED14	0x00000200
System Handlers 4-7 Priority Register	Read/write	0xE000ED18	0x00000000
System Handlers 8-11 Priority Register	Read/write	0xE000ED1C	0x00000000
System Handlers 12-15 Priority Register	Read/write	0xE000ED20	0x00000000
System Handler Control and State Register	Read/write	0xE000ED24	0x00000000
Configurable Fault Status Registers	Read/write	0xE000ED28	0x00000000
Hard Fault Status Register	Read/write	0xE000ED2C	0x00000000
Debug Fault Status Register	Read/write	0xE000ED30	0x00000000
Mem Manage Address Register	Read/write	0xE000ED34	Unpredictable
Bus Fault Address Register	Read/write	0xE000ED38	Unpredictable
Auxiliary Fault Status Register	Read/write	0xE000ED3C	0x00000000
PFR0: Processor Feature register0	Read-only	0xE000ED40	0x00000030
PFR1: Processor Feature register1	Read-only	0xE000ED44	0x00000200
DFR0: Debug Feature register0	Read-only	0xE000ED48	0x00100000
AFR0: Auxiliary Feature register0	Read-only	0xE000ED4C	0x00000000
MMFR0: Memory Model Feature register0	Read-only	0xE000ED50	0x00000030
MMFR1: Memory Model Feature register1	Read-only	0xE000ED54	0x00000000
MMFR2: Memory Model Feature register2	Read-only	0xE000ED58	0x00000000
MMFR3: Memory Model Feature register3	Read-only	0xE000ED5C	0x00000000
ISAR0: ISA Feature register0	Read-only	0xE000ED60	0x01141110
ISAR1: ISA Feature register1	Read-only	0xE000ED64	0x02111000
ISAR2: ISA Feature register2	Read-only	0xE000ED68	0x21112231
ISAR3: ISA Feature register3	Read-only	0xE000ED6C	0x01111110

Table 3-1 NVIC registers (continued)

Name of register	Type	Address	Reset value
ISAR4: ISA Feature register ⁴	Read-only	0xE000ED70	0x01310102
Software Trigger Interrupt Register	Write Only	0xE000EF00	-
Peripheral identification register (PID4)	Read-only	0xE000EFD0	0x04
Peripheral identification register (PID5)	Read-only	0xE000EFD4	0x00
Peripheral identification register (PID6)	Read-only	0xE000EFD8	0x00
Peripheral identification register (PID7)	Read-only	0xE000EFD8	0x00
Peripheral identification register Bits [7:0] (PID0)	Read-only	0xE000EFE0	0x00
Peripheral identification register Bits [15:8] (PID1)	Read-only	0xE000EFE4	0xB0
Peripheral identification register Bits [23:16] (PID2)	Read-only	0xE000EFE8	0x2B
Peripheral identification register Bits [31:24] (PID3)	Read-only	0xE000EFEC	0x00
Component identification register Bits [7:0] (CID0)	Read Only	0xE000EFF0	0x0D
Component identification register Bits [15:8] (CID1)	Read-only	0xE000EFF4	0xE0
Component identification register Bits [23:16] (CID2)	Read-only	0xE000EFF8	0x05
Component identification register Bits [31:24] (CID3)	Read-only	0xE000EFFF	0xB1

a. Reset value depends on the number of interrupts defined.

3.1.2 Core debug registers

Table 3-2 gives a summary of the core debug registers. For a detailed description of the core debug registers, see Chapter 10 *Core Debug*.

Table 3-2 Core debug registers

Name of register	Type	Address	Reset Value
Debug Halting Control and Status Register	Read/Write	0xE000EDF0	0x00000000 ^a
Debug Core Register Selector Register	Write-only	0xE000EDF4	-
Debug Core Register Data Register	Read/Write	0xE000EDF8	-
Debug Exception and Monitor Control Register.	Read/Write	0xE000EDFC	0x00000000 ^b

- a. Bits [5], [3], [2], [1], [0] are reset by **PORESETn**. Bit [1] is also reset by **SYSRESETn** and by writing a 1 to the **VECTRESET** bit of the Application Interrupt and Reset Control Register.
- b. Bits [16], [17], [18], [19] are also reset by **SYSRESETn** and by writing a 1 to the **VECTRESET** bit of the Application Interrupt and Reset Control Register.

3.1.3 System debug registers

This section lists the system debug registers.

Flash Patch and Breakpoint registers

Table 3-3 gives a summary of the *Flash Patch and Breakpoint* (FPB) registers. For a detailed description of the FPB registers, see Chapter 11 *System Debug*.

Table 3-3 Flash patch register summary

Name	Type	Address	Reset value	Description
FP_CTRL	Read/write	0xE0002000	Bit [0] is reset to 1'b0	Flash Patch Control Register
FP_REMAP	Read/write	0xE0002004	-	Flash Patch Remap Register
FP_COMP0	Read/write	0xE0002008	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP1	Read/write	0xE000200C	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP2	Read/write	0xE0002010	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP3	Read/write	0xE0002014	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP4	Read/write	0xE0002018	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP5	Read/write	0xE000201C	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP6	Read/write	0xE0002020	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
FP_COMP7	Read/write	0xE0002024	Bit [0] is reset to 1'b0	Flash Patch Comparator Registers
PID4	Read-only	0xE0002FD0	-	Value 0x04
PID5	Read-only	0xE0002FD4	-	Value 0x00
PID6	Read-only	0xE0002FD8	-	Value 0x00
PID7	Read-only	0xE0002FDC	-	Value 0x00
PID0	Read-only	0xE0002FE0	-	Value 0x03
PID1	Read-only	0xE0002FE4	-	Value 0xB0

Table 3-3 Flash patch register summary (continued)

Name	Type	Address	Reset value	Description
PID2	Read-only	0xE0002FE8	-	Value 0x2B
PID3	Read-only	0xE0002FEC	-	Value 0x00
CID0	Read-only	0xE0002FF0	-	Value 0x0D
CID1	Read-only	0xE0002FF4	-	Value 0xE0
CID2	Read-only	0xE0002FF8	-	Value 0x05
CID3	Read-only	0xE0002FFC	-	Value 0xB1

Data Watchpoint and Trace registers

Table 3-4 gives a summary of the *Data Watchpoint and Trace* (DWT) registers. For a detailed description of the DWT registers, see Chapter 11 *System Debug*.

Table 3-4 DWT register summary

Name	Type	Address	Reset value	Description
DWT_CTRL	Read/write	0xE0001000	0x40000000	DWT Control Register
DWT_CYCCNT	Read/write	0xE0001004	0x00000000	DWT Current PC Sampler Cycle Count Register
DWT_CPICNT	Read/write	0xE0001008	-	DWT Current CPI Count Register
DWT_EXCCNT	Read/write	0xE000100C	-	DWT Current Interrupt Overhead Count Register
DWT_SLEPCNT	Read/write	0xE0001010	-	DWT Current Sleep Count Register
DWT_LSUCNT	Read/write	0xE0001014	-	DWT Current LSU Count Register
DWT_FOLDCNT	Read/write	0xE0001018	-	DWT Current Fold Count Register
DWT_PCSR	Read-only	0xE000101C	-	DWT PC Sample Register
DWT_COMP0	Read/write	0xE0001020	-	DWT Comparator Register
DWT_MASK0	Read/write	0xE0001024	-	DWT Mask Registers
DWT_FUNCTION0	Read/write	0xE0001028	0x00000000	DWT Function Registers
DWT_COMP1	Read/write	0xE0001030	-	DWT Comparator Register

Table 3-4 DWT register summary (continued)

Name	Type	Address	Reset value	Description
DWT_MASK1	Read/write	0xE0001034	-	DWT Mask Registers
DWT_FUNCTION1	Read/write	0xE0001038	0x00000000	DWT Function Registers
DWT_COMP2	Read/write	0xE0001040	-	DWT Comparator Register
DWT_MASK2	Read/write	0xE0001044	-	DWT Mask Registers
DWT_FUNCTION2	Read/write	0xE0001048	0x00000000	DWT Function Registers
DWT_COMP3	Read/write	0xE0001050	-	DWT Comparator Register
DWT_MASK3	Read/write	0xE0001054	-	DWT Mask Registers
DWT_FUNCTION3	Read/write	0xE0001058	0x00000000	DWT Function Registers
PID4	Read-only	0xE0001FD0	0x04	Value 0x04
PID5	Read-only	0xE0001FD4	0x00	Value 0x00
PID6	Read-only	0xE0001FD8	0x00	Value 0x00
PID7	Read-only	0xE0001FDC	0x00	Value 0x00
PID0	Read-only	0xE0001FE0	0x02	Value 0x02
PID1	Read-only	0xE0001FE4	0xB0	Value 0xB0
PID2	Read-only	0xE0001FE8	0x0B0	Value 0x2B
PID3	Read-only	0xE0001FEC	0x00	Value 0x00
CID0	Read-only	0xE0001FF0	0x0D	Value 0x0D
CID1	Read-only	0xE0001FF4	0xE0	Value 0xE0
CID2	Read-only	0xE0001FF8	0x05	Value 0x05
CID3	Read-only	0xE0001FFC	0xB1	Value 0xB1

Instrumentation Trace Macrocell registers

Table 3-5 gives a summary of the *Instrumentation Trace Macrocell* (ITM) registers. For a detailed description of the ITM registers, see Chapter 11 *System Debug*

Table 3-5 ITM register summary

Name	Type	Address	Reset value
Stimulus Ports 0-31	Read/write	0xE0000000-0xE000007C	-
Trace Enable	Read/write	0xE0000E00	0x00000000
Trace Privilege	Read/write	0xE0000E40	0x00000000
Trace Control Register	Read/write	0xE0000E80	0x00000000
Integration Write	Write-only	0xE0000EF8	0x00000000
Integration Read	Read-only	0xE0000EFC	0x00000000
Integration Mode Control	Read/write	0xE0000F00	0x00000000
Lock Access Register	Write-only	0xE0000FB0	0x00000000
Lock Status Register	Read-only	0xE0000FB4	0x00000003
PID4	Read-only	0xE000FD0	0x00000004
PID5	Read-only	0xE000FD4	0x00000000
PID6	Read-only	0xE000FD8	0x00000000
PID7	Read-only	0xE000FDC	0x00000000
PID0	Read-only	0xE000FE0	0x00000001
PID1	Read-only	0xE000FE4	0x000000B0
PID2	Read-only	0xE000FE8	0x0000002B
PID3	Read-only	0xE000FEC	0x00000000
CID0	Read-only	0xE000FF0	0x0000000D
CID1	Read-only	0xE000FF4	0x000000E0
CID2	Read-only	0xE000FF8	0x00000005
CID3	Read-only	0xE000FFC	0x000000B1

Advanced High Performance Bus Access Port registers

Table 3-6 gives a summary of the *Advanced High-performance Bus Access Port* (AHB-AP) registers. For a detailed description of the AHB-AP registers, see Chapter 11 *System Debug*.

Table 3-6 AHB-AP register summary

Name	Type	Address	Reset value
Control and Status Word	Read/write	0x00	See Register
Transfer Address	Read/write	0x04	-
Data Read/write	Read/write	0x0C	-
Banked Data 0	Read/write	0x10	-
Banked Data 1	Read/write	0x14	-
Banked Data 2	Read/write	0x18	-
Banked Data 3	Read/write	0x1C	-
Debug ROM Address	Read-only	0xF8	0xE000E000
Identification Register	Read-only	0xFC	0x24770011

3.1.4 Debug interface port registers

Table 3-7 gives a summary of the debug interface port registers. For a detailed description of the debug interface port registers, see Chapter 13 *Debug Port*.

Table 3-7 Summary of Debug interface port registers

Name	SWJ-DP	SW-DP	Description
ABORT	Yes	Yes	The Abort Register
IDCODE	Yes	Yes	The Identification Code Register
CTRL/STAT	Yes	Yes	The Control/Status Register
SELECT	Yes	Yes	The AP Select Register
RDBUFF	Yes	Yes	The Read Buffer Register

Table 3-7 Summary of Debug interface port registers (continued)

Name	SWJ-DP	SW-DP	Description
WCR	No	Yes	The Wire Control Register
RESEND	No	Yes	The Read Resend Register

3.1.5 Memory Protection Unit registers

Table 3-8 gives a summary of the *Memory Protection Unit* (MPU) registers. For a detailed description of the MPU registers, see Chapter 9 *Memory Protection Unit*.

Table 3-8 MPU registers

Name	Type	Address	Reset value
MPU Type Register	Read Only	0xE000ED90	0x00000800
MPU Control Register	Read/Write	0xE000ED94	0x00000000
MPU Region Number register	Read/Write	0xE000ED98	-
MPU Region Base Address register	Read/Write	0xE000ED9C	-
MPU Region Attribute and Size registers	Read/Write	0xE000EDA0	-
MPU Alias 1 Region Base Address register	Alias of D9C	0xE000EDA4	-
MPU Alias 1 Region Attribute and Size register	Alias of DA0	0xE000EDA8	-
MPU Alias 2 Region Base Address register	Alias of D9C	0xE000EDAC	-
MPU Alias 2 Region Attribute and Size register	Alias of DA0	0xE000EDB0	-
MPU Alias 3 Region Base Address register	Alias of D9C	0xE000EDB4	-
MPU Alias 3 Region Attribute and Size register	Alias of DA0	0xE000EDB8	-

3.1.6 Trace Port Interface Unit registers

Table 3-9 gives a summary of the *Trace Port Interface Unit* (TPIU) registers. For a detailed description of the TPIU registers, see Chapter 17 *Trace Port Interface Unit*.

Table 3-9 TPIU registers

Name of register	Type	Address	Reset value
Supported Sync Port Sizes Register	Read-only	0xE0040000	0bxx0x
Current Sync Port Size Register	Read/write	0xE0040004	0x01
Async Clock Prescaler Register	Read/write	0xE0040010	0x0000
Selected Pin Protocol Register	Read/write	0xE00400F0	0x01
Formatter and Flush Status Register	Read-only	0xE0040300	0x08
Formatter and Flush Control Register	Read/write	0xE0040304	0x00 or 0x102
Formatter Synchronization Counter Register	Read-only	0xE0040308	0x00
Integration Register: ITATBCTR2	Read-only	0xE0040EF0	0x0
Integration Register: ITATBCTR0	Read-only	0xE0040EF8	0x0
Integration Mode Control Register	Read/write	0xE0040F00	0x0
Integration register : FIFO data 0	Read only	0xE0040EEC	0x--000000
Integration register : FIFO data 1	Read only	0xE0040EFC	0x--000000
Claim tag set register	Read/write	0xE0040FA0	0xF
Claim tag clear register	Read/write	0xE0040FA4	0x0
Device ID register	Read only	0xE0040FCC	0x11
PID4	Read only	0xE0040FD0	0x04
PID5	Read only	0xE0040FD4	0x00
PID6	Read only	0xE0040FD8	0x00
PID7	Read only	0xE0040FDC	0x00
PID0	Read only	0xE0040FE0	0x23
PID1	Read only	0xE0040FE4	0xB9
PID2	Read only	0xE0040FE8	0x2B

Table 3-9 TPIU registers (continued)

Name of register	Type	Address	Reset value
PID3	Read only	0xE0040FEC	0x00
CID0	Read only	0xE0040FF0	0x0D
CID1	Read only	0xE0040FF4	0x90
CID2	Read only	0xE0040FF8	0x05
CID3	Read only	0xE0040FFC	0xB1

3.1.7 Embedded Trace Macrocell registers

Table 3-10 gives a summary of the *Embedded Trace Macrocell* (ETM) registers. For a detailed description of the ETM registers, see Chapter 14 *Embedded Trace Macrocell*.

Table 3-10 ETM registers

Name	Type	Address	Present
ETM Control	Read/write	0xE0041000	Yes
Configuration Code	Read-only	0xE0041004	Yes
Trigger event	Write-only	0xE0041008	Yes
ASIC Control	Write-only	0xE004100C	No
ETM Status	Read-only or read/write	0xE0041010	Yes
System Configuration	Read-only	0xE0041014	Yes
TraceEnable	Write-only	0xE0041018, 0xE004101C	No
TraceEnable Event	Write-only	0xE0041020	Yes
TraceEnable Control 1	Write-only	0xE0041024	Yes
FIFOFULL Region	Write-only	0xE0041028	No
FIFOFULL Level	Write-only or read/write	0xE004102C	Yes
ViewData	Write-only	0xE0041030-0xE004103C	No
Address Comparators	Write-only	0xE0041040- 0xE004113C	No
Counters	Write-only	0xE0041140-0xE004157C	No

Table 3-10 ETM registers (continued)

Name	Type	Address	Present
Sequencer	Read/write	0xE0041180- 0xE0041194, 0xE0041198	No
External Outputs	Write-only	0xE00411A0-0xE00411AC	No
CID Comparators	Write-only	0xE00411B0-0xE00411BC	No
Implementation specific	Write-only	0xE00411C0-0xE00411DC	No
Synchronization Frequency	Read-only	0xE00411E0	Yes
ETM ID	Read-only	0xE00411E4	Yes
Configuration Code Extension	Read-only	0xE00411E8	Yes
Extended External Input Selector	Write-only	0xE00411EC	No
TraceEnable Start/Stop Embedded ICE	Read/write	0xE00411F0	Yes
Embedded ICE Behavior Control	Write-only	0xE00411F4	No
CoreSight Trace ID	Read/write	0xE0041200	Yes
OS Save/Restore	Write-only	0xE0041304-0xE0041308	No
ITMISCIN	Read-only	0xE0041EE0	Yes
ITTRIGOUT	Write-only	0xE0041EE8	Yes
ITATBCTR2	Read-only	0xE0041EF0	Yes
ITATBCTR0	Write-only	0xE0041EF8	Yes
Integration Mode Control	Read/write	0xE0041F00	Yes
Claim Tag	Read/write	0xE0041FA0-0xE0041FA4	Yes
Lock Access	Write-only	0xE0041FB0-0xE0041FB4	Yes
Authentication Status	Read-only	0xE0041FB8	Yes
Device Type	Read-only	0xE0040FCC	Yes
Peripheral ID 4	Read-only	0xE0041FD0	Yes
Peripheral ID 5	Read-only	0xE0041FD4	Yes
Peripheral ID 6	Read-only	0xE0041FD8	Yes
Peripheral ID 7	Read-only	0xE0041FDC	Yes

Table 3-10 ETM registers (continued)

Name	Type	Address	Present
Peripheral ID 0	Read-only	0xE0041FE0	Yes
Peripheral ID 1	Read-only	0xE0041FE4	Yes
Peripheral ID 2	Read-only	0xE0041FE8	Yes
Peripheral ID 3	Read-only	0xE0041FEC	Yes
Component ID 0	Read-only	0xE0041FF0	Yes
Component ID 1	Read-only	0xE0041FF4	Yes
Component ID 2	Read-only	0xE0041FF8	Yes
Component ID 3	Read-only	0xE0041FFC	Yes

Chapter 4

Memory Map

This chapter describes the processor fixed memory map and its bit-banding feature. It contains the following sections:

- *About the memory map* on page 4-2
- *Bit-banding* on page 4-5
- *ROM memory table* on page 4-7.

4.1 About the memory map

Figure 4-1 shows the fixed memory map.

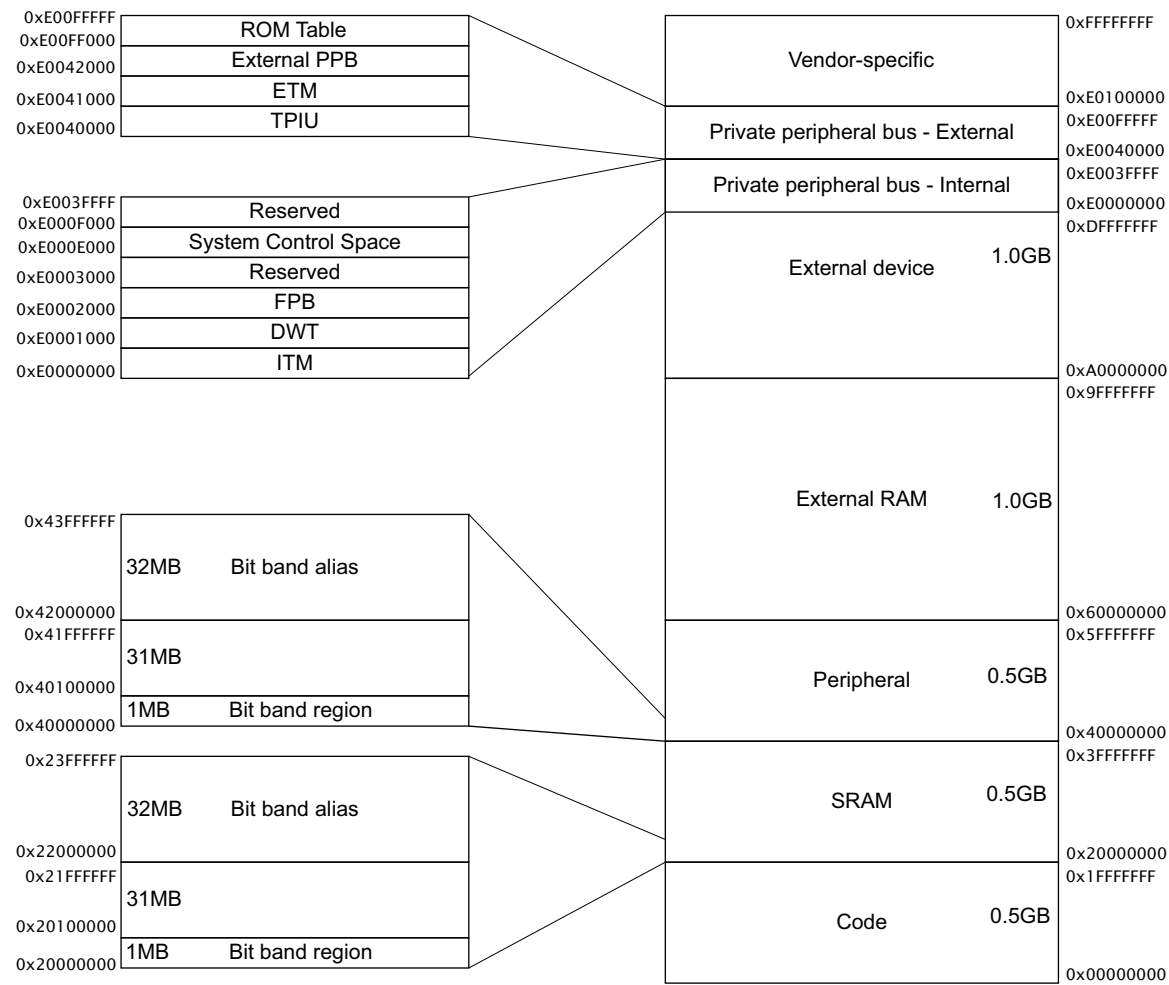


Figure 4-1 Processor memory map

Table 4-1 shows the processor interfaces that are addressed by the different memory map regions

Table 4-1 Memory interfaces

Memory Map	Interface
Code	Instruction fetches are performed over the ICode bus. Data accesses are performed over the DCode bus.
SRAM	Instruction fetches and data accesses are performed over the system bus.
SRAM_bitband	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
Peripheral	Instruction fetches and data accesses are performed over the system bus.
Periph_bitband	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
External RAM	Instruction fetches and data accesses are performed over the system bus.
External Device	Instruction fetches and data accesses are performed over the system bus.
Private Peripheral Bus	<p>Accesses to:</p> <ul style="list-style-type: none"> • <i>Instrumentation Trace Macrocell (ITM)</i> • <i>Nested Vectored Interrupt Controller (NVIC)</i> • <i>Flashpatch and Breakpoint (FPB)</i> • <i>Data Watchpoint and Trace (DWT)</i> • <i>Memory Protection Unit (MPU)</i> <p>are performed to the processor internal <i>Private Peripheral Bus (PPB)</i>.</p> <p>Accesses to:</p> <ul style="list-style-type: none"> • <i>Trace Point Interface Unit (TPIU)</i> • <i>Embedded Trace Macrocell (ETM)</i> • System areas of the PPB memory map <p>are performed over the external PPB interface.</p> <p>This memory region is <i>Execute Never (XN)</i>, and so instruction fetches are prohibited. An MPU, if present, cannot change this.</p>
System	System segment for vendor system peripherals. This memory region is XN, and so instruction fetches are prohibited. An MPU, if present, cannot change this.

Table 4-2 shows the permissions of the processor memory regions.

Table 4-2 Memory region permissions

Name	Region	Device type	XN	Cache
Code	0x00000000-0x1FFFFFFF	Normal	-	WT
SRAM	0x20000000-0x3FFFFFFF	Normal	-	WBWA
SRAM_1M	+0000000	-	-	-
SRAM_31M	+0100000	-	-	-
SRAM_bitband	+2000000	Internal	-	-
SRAM	+4000000	-	-	-
Peripheral	0x40000000-0x5FFFFFFF	Device	XN	-
Periph_1IM	+0000000	-	-	-
Periph_31IM	+0100000	-	-	-
Periph_bit band	+2000000	Internal	-	-
Peripheral	+4000000	-	-	-
External RAM	0x60000000-0x7FFFFFFF	Normal	-	WBWA
External RAM	0x80000000-0x9FFFFFFF	Normal	-	WT
External Device	0xA0000000-0xBFFFFFFF	Device	XN	-
External Device	0xC0000000-0xDFFFFFFF	Device	XN	-
System	0xE0000000-0xFFFFFFFF	-	XN	-
Private Peripheral Bus	+0000000	SO, shared	XN	-
Vendor_SYS	+0100000	Device	XN	-

Note

Private Peripheral Bus and System space at 0xE0000000 - 0xFFFFFFFF are permanently XN. The MPU cannot change this.

For a description of the processor bus interfaces, see Chapter 12 *Bus Interface*.

4.2 Bit-banding

The processor memory map includes two bit-band regions. These occupy the lowest 1MB of the SRAM and Peripheral memory regions respectively. These bit-band regions map each word in an alias region of memory to a bit in a bit-band region of memory.

The memory map has two 32-MB alias regions that map to two 1-MB bit-band regions:

- Accesses to the 32-MB SRAM alias region map to the 1-MB SRAM bit-band region.
- Accesses to the 32-MB peripheral alias region map to the 1-MB peripheral bit-band region.

A mapping formula shows how to reference each word in the alias region to a corresponding bit, or target bit, in the bit-band region. The mapping formula is:

$$\text{bit_word_offset} = (\text{byte_offset} \times 32) + (\text{bit_number} \times 4)$$

$$\text{bit_word_addr} = \text{bit_band_base} + \text{bit_word_offset}$$

where:

- Bit_word_offset is the position of the target bit in the bit-band memory region.
- Bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
- Bit_band_base is the starting address of the alias region.
- Byte_offset is the number of the byte in the bit-band region that contains the targeted bit.
- Bit_number is the bit position (0-7) of the targeted bit.

Figure 4-2 on page 4-6 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FFFE0 maps to bit [0] of the bit-band byte at 0x200FFFFF:
 $0x23FFFE0 = 0x22000000 + (0xFFFF \times 32) + 0 \times 4.$
- The alias word at 0x23FFFFC maps to bit [7] of the bit-band byte at 0x200FFFFF:
 $0x23FFFFC = 0x22000000 + (0xFFFF \times 32) + 7 \times 4.$
- The alias word at 0x2200000 maps to bit [0] of the bit-band byte at 0x2000000:
 $0x2200000 = 0x22000000 + (0 \times 32) + 0 \times 4.$
- The alias word at 0x220001C maps to bit [7] of the bit-band byte at 0x2000000:
 $0x220001C = 0x22000000 + (0 \times 32) + 7 \times 4.$

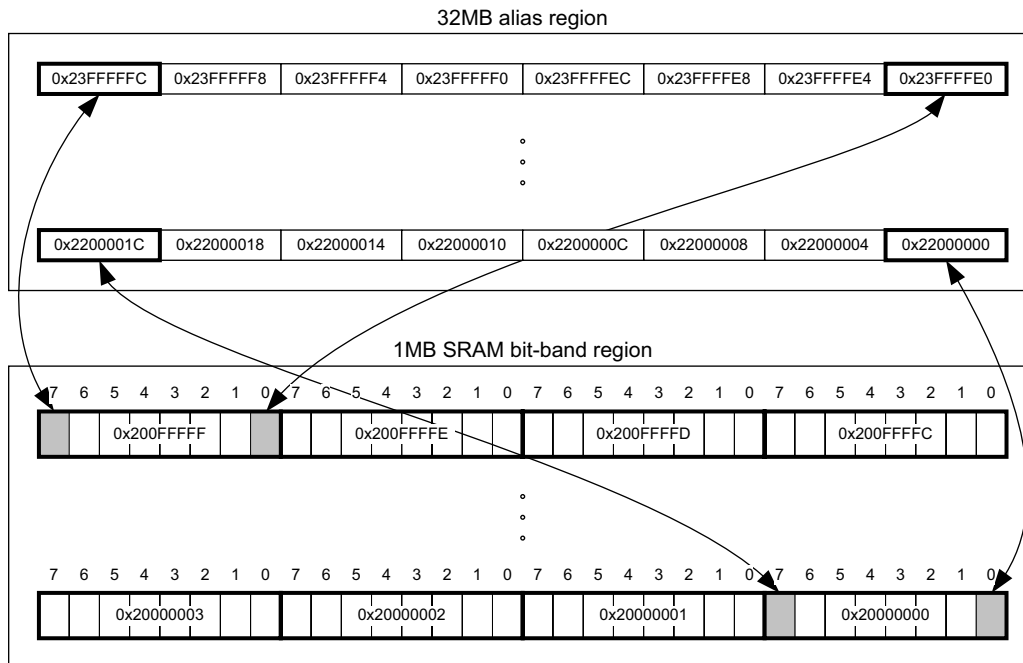


Figure 4-2 Bit-band mapping

4.2.1 Directly accessing an alias region

Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

Bit [0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit [0] set writes a 1 to the bit-band bit, and writing a value with bit [0] cleared writes a 0 to the bit-band bit.

Bits [31:1] of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

Reading a word in the alias region returns either 0x01 or 0x00. A value of 0x01 indicates that the targeted bit in the bit-band region is set. A value of 0x00 indicates that the targeted bit is clear. Bits [31:1] are zero.

4.2.2 Directly accessing a bit-band region

You can directly access the bit-band region with normal reads and writes, and writes to that region.

4.3 ROM memory table

Table 4-3 describes the ROM memory.

Table 4-3 ROM table

Offset	Value	Name	Description
0x000	0xFFFF0F03	NVIC	Points to the NVIC at 0xE000E000.
0x004	0xFFFF0202 or 003 if present	DWT	Points to the Data Watchpoint and Trace block at 0xE0001000. Value has bit [0] set if DWT is present.
0x008	0xFFFF0302 or 003 if present	FPB	Points to the Flash Patch and Breakpoint block at 0xE0002000. Value has bit [0] set to 1 if FPB is present.
0x00C	0xFFFF0102 or 003 if present	ITM	Points to the Instrumentation Trace block at 0xE0000000. Value has bit [0] set if ITM is present.
0x010	0xFFFF4102 or 003 if present	TPIU	Points to the TPIU. Value has bit [0] set to 1 if TPIU is present. TPIU is at 0xE0040000.
0x014	0xFFFF4202 or 003 if present	ETM	Points to the ETM. Value has bit [0] set to 1 if ETM is present. ETM is at 0xE0041000.
0x018	0	End	Marks the end of the ROM table. If CoreSight components are added, they are added starting from this location and the End marker is moved to the next location after the additional components.
0xFCC	0x1	MEMTYPE	Bits [31:1] RAZ. Bit [0] is set when the system memory map is accessible using the DAP. Bit [0] is clear when only debug resources are accessible using the DAP.
0xFD0	0x0	PID4	-
0xFD4	0x0	PID5	-
0xFD8	0x0	PID6	-
0xFDC	0x0	PID7	-
0xFE0	0x0	PID0	-
0xFE4	0x0	PID1	-
0xFE8	0x0	PID2	-
0xFEC	0x0	PID3	-
0xFF0	0x0D	CID0	-

Table 4-3 ROM table (continued)

Offset	Value	Name	Description
0xFF4	0x10	CID1	-
0xFF8	0x05	CID2	-
0xFFC	0xB1	CID3	-

Chapter 5

Exceptions

This chapter describes the exception model of the processor. It contains the following sections:

- *About the exception model* on page 5-2
- *Exception types* on page 5-4
- *Exception priority* on page 5-6
- *Privilege and stacks* on page 5-9
- *Pre-emption* on page 5-11
- *Tail-chaining* on page 5-14
- *Late-arriving* on page 5-15
- *Exit* on page 5-17
- *Resets* on page 5-20
- *Exception control transfer* on page 5-24
- *Setting up multiple stacks* on page 5-25
- *Abort model* on page 5-27
- *Activation levels* on page 5-32
- *Flowcharts* on page 5-34.

5.1 About the exception model

The processor and the *Nested Vectored Interrupt Controller* (NVIC) prioritize and handle all exceptions. All exceptions are handled in Handler mode. Processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the *Interrupt Service Routine* (ISR). The vector is fetched in parallel to the state saving, enabling efficient interrupt entry. The processor supports tail-chaining that enables back-to-back interrupts without the overhead of state saving and restoration. The following features enable efficient, low latency exception handling:

- Automatic state saving and restoring. The processor pushes state registers on the stack before entering the ISR, and pops them after exiting the ISR with no instruction overhead.
- Automatic reading of the vector table entry that contains the ISR address in code memory or data SRAM. This is performed in parallel to the state saving.

Note

Vector table entries are ARM/Thumb interworking compatible.

This causes bit [0] of the vector value to load into the EPSR T-bit on exception entry. Creating a table entry with bit [0] clear generates an INVSTATE fault on the first instruction of the handler corresponding to this vector.

- Support for tail-chaining. In tail-chaining, the processor handles back-to-back interrupts without popping and pushing registers between ISRs.
- Dynamic reprioritization of interrupts.
- Closely-coupled interface between the processor core and the NVIC to enable early processing of interrupts and processing of late-arriving interrupts with higher priority.
- Configurable number of interrupts, from 1 to 240.
- Configurable number of interrupt priorities, from 3 to 8 bits (8 to 256 levels).
- Separate stacks and privilege levels for Handler and Thread modes.
- ISR control transfer using the calling conventions of the C/C++ standard *ARM Architecture Procedure Call Standard* (AAPCS).
- Priority masking to support critical regions.

Note

The number of interrupts, and bits of interrupt priority, are configured during implementation. Software can choose only to enable a subset of the configured number of interrupts, and can choose how many bits of the configured priorities to use.

5.2 Exception types

Various types of exceptions exist in the processor. A fault is an exception that results from an error condition because of instruction execution. Faults can be reported synchronously or asynchronously to the instruction that caused them. In general, faults are reported synchronously. The Imprecise Bus Fault is an asynchronous fault supported in the ARMv7-M profile. A synchronous fault is always reported with the instruction that caused the fault. An asynchronous fault does not guarantee how it is reported with respect to the instruction that caused the fault.

For more information on exceptions, see the *ARMv7-M Architecture Reference Manual*.

Table 5-1 shows the exception type, position, and priority. Position refers to the word offset from the start of the vector table. The lower numbers shown in the Priority column of the table are higher priority. How the types are activated, synchronously or asynchronously, is also shown. The exact meaning and use of priorities is explained in *Exception priority* on page 5-6.

Table 5-1 Exception types

Exception type	Position	Priority	Description
-	0	-	Stack top is loaded from first entry of vector table on reset.
Reset	1	-3 (highest)	Invoked on power up and warm reset. On first instruction, drops to lowest priority (Thread mode). This is asynchronous.
Non-maskable Interrupt	2	-2	Cannot be stopped or pre-empted by any exception but reset. This is asynchronous.
Hard Fault	3	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled. This is synchronous.
Memory Management	4	Configurable ^a	<i>Memory Protection Unit</i> (MPU) mismatch, including access violation and no match. This is synchronous. This is used even if the MPU is disabled or not present, to support the <i>Executable Never</i> (XN) regions of the default memory map.
Bus Fault	5	Configurable ^a	Pre-fetch fault, memory access fault, and other address/memory related. This is synchronous when precise and asynchronous when imprecise.
Usage Fault	6	Configurable ^a	Usage fault, such as Undefined instruction executed or illegal state transition attempt. This is synchronous.

Table 5-1 Exception types (continued)

Exception type	Position	Priority	Description
-	7-10	-	Reserved
SVCall	11	Configurable	System service call with SVC instruction. This is synchronous.
Debug Monitor	12	Configurable	Debug monitor, when not halting. This is synchronous, but only active when enabled. It does not activate if lower priority than the current activation.
-	13	-	Reserved
PendSV	14	Configurable	Pendable request for system service. This is asynchronous and only pended by software.
SysTick	15	Configurable	System tick timer has fired. This is asynchronous.
External Interrupt	16 and more	Configurable	Asserted from outside the core, INTISR[239:0] , and fed through the NVIC (prioritized). These are all asynchronous.

- a. You can change the priority of this exception. See *System Handler Priority Registers bit assignments* on page 8-29. *Settable* is an NVIC priority value of 0 to N, where N is the largest priority value implemented. Internally, the highest user-settable priority (0) is treated as 4. You can enable or disable this fault. See *System Handler Control and State Register bit assignments* on page 8-30.

5.3 Exception priority

Table 5-2 shows how priority affects when and how the processor takes an exception. It lists the actions an exception can take based on priority.

Table 5-2 Priority-based actions of exceptions

Action	Description
Pre-emption	<p>New exception has higher priority than current exception priority or thread and interrupts current flow.</p> <p>This is the response to a pended interrupt, causing entry to an ISR if the pended interrupt is higher priority than the active ISR or thread. When one ISR pre-empts another, the interrupts are nested.</p> <p>On exception entry the processor automatically saves processor state, which is pushed on to the stack. In parallel with this, the vector corresponding to the interrupt is fetched. Execution of the first instruction of the ISR starts when processor state is saved and the first instruction of the ISR enters the execute stage of the processor pipeline. The state saving is performed over the System bus and DCode bus. The vector fetch is performed over either the System bus or the ICode bus depending on where the vector table is located, see <i>Vector Table Offset Register</i> on page 8-21.</p>
Tail-chain	<p>A mechanism used by the processor to speed up interrupt servicing. On completion of an ISR, if there is a pending interrupt of higher priority than the ISR or thread that is being returned to, the stack pop is skipped and control is transferred to the new ISR.</p>
Return	<p>With no pending exceptions or no pending exceptions with higher priority than a stacked ISR, the processor pops the stack and returns to stacked ISR or Thread Mode.</p> <p>On completion of an ISR the processor automatically restores the processor state by popping the stack to the state prior to the interrupt that caused the ISR to be entered. If a new interrupt arrives during the state restoration, and that interrupt is of higher priority than the ISR or thread that is being returned to, then the state restoration is abandoned and the new interrupt is handled as a tail-chain.</p>
Late-arriving	<p>A mechanism used by the processor to speed up pre-emption. If a higher priority interrupt arrives during state saving for a previous pre-emption, the processor switches to handling the higher priority interrupt instead and initiates the vector fetch for that interrupt. The state saving is not effected by late arrival because the state saved is the same for both interrupts, and the state saving continues uninterrupted. Late arriving interrupts are managed until the first instruction of the ISR enters the execute stage of the processor pipeline. On return, the normal tail-chaining rules apply.</p>

In the processor exception model, priority determines when and how the processor takes exceptions. You can:

- assign software priority levels to interrupts
- group priorities by splitting priority levels into pre-emption priorities and subpriorities.

5.3.1 Priority levels

The NVIC supports software-assigned priority levels. You can assign a priority level from 0 to 255 to an interrupt by writing to the eight-bit PRI_N field in an Interrupt Priority Register, see *Interrupt Priority Registers* on page 8-17. Hardware priority decreases with increasing interrupt number. Priority level 0 is the highest priority level, and priority level 255 is the lowest. The priority level overrides the hardware priority. For example, if you assign priority level 1 to **IRQ[0]** and priority level 0 to **IRQ[31]**, then **IRQ[31]** has higher priority than **IRQ[0]**.

———— **Note** —————

Software prioritization does not affect reset, *Non-Maskable Interrupt* (NMI), and hard fault. They always have higher priority than the external interrupts.

When multiple interrupts have the same priority number, the pending interrupt with the lowest interrupt number takes precedence. For example, if both **IRQ[0]** and **IRQ[1]** are priority level 1, then **IRQ[0]** has higher priority than **IRQ[1]**.

For more information on the PRI_N fields, see *Interrupt Priority Registers* on page 8-17.

5.3.2 Priority grouping

To increase priority control in systems with large numbers of interrupts, the NVIC supports priority grouping. You can use the PRIGROUP field in the *Application Interrupt and Reset Control Register* on page 8-22 to split the value in every PRI_N field into a pre-emption priority field and a subpriority field. The pre-emption priority group is referred to as the group priority. Where multiple pending exceptions share the same group priority, the sub-priority bit field resolves the priority within a group. This is referred to as the sub-priority within the group. The combination of the group priority and the sub-priority is referred to generally as the priority. Where two pending exceptions have the same priority, the lower pending exception number has priority over the higher pending exception number. This is consistent with the priority precedence scheme.

Table 5-3 shows how writing to PRIGROUP splits an eight bit PRI_N field into a pre-emption priority field (x) and a subpriority field (y).

Table 5-3 Priority grouping

Interrupt priority level field, PRI_N[7:0]					
PRIGROUP[2:0]	Binary point position	Pre-emption field	Subpriority field	Number of pre-emption priorities	Number of subpriorities
b000	bxxxxxxx.y	[7:1]	[0]	128	2
b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
b110	bx.yyyyyyy	[7]	[6:0]	2	128
b111	b.yyyyyyyy	None	[7:0]	0	256

———— **Note** ————

- Table 5-3 shows the priorities for the processor configured with eight bits of priority.
- For a processor configured with less than eight bits of priority, the lower bits of the register are always 0. For example, if four bits of priority are implemented, **PRI_N[7:4]** sets the priority, and **PRI_N[3:0]** is 4'b0000.

An interrupt can pre-empt another interrupt in progress only if its pre-emption priority is higher than that of the interrupt in progress.

For more information on priority optimizations, priority level grouping, and priority masking, see the *ARMv7-M Architecture Reference Manual*.

5.4 Privilege and stacks

The processor supports two separate stacks:

Process stack

You can configure Thread mode to use the process stack. Thread mode uses the main stack out of reset. SP_process is the *Stack Pointer* (SP) register for the process stack.

Main stack

Handler mode uses the main stack. SP_main is the SP register for the main stack.

Only one stack, the process stack or the main stack, is visible at any time. After pushing the eight registers, the ISR uses the main stack, and all subsequent interrupt pre-emptions use the main stack. The stack that saves context is as follows:

- Thread mode uses either the main stack or the process stack, depending on the value of the CONTROL bit [1] that *Move to Status Register* (MSR) or *Move to Register from Status* (MRS) can access. Appropriate EXC_RETURN values can also set this bit when exiting an ISR. An exception that pre-empts a user thread saves the context of the user thread on the stack that the Thread mode is using.
- All exceptions use the main stack for their own local variables.

Using the process stack for the Thread mode and the main stack for exceptions supports *Operating System* (OS) scheduling. To reschedule, the kernel only requires to save the eight registers not pushed by hardware, r4-r11, and to copy SP_process into the *Thread Control Block* (TCB). If the processor saved the context on the main stack, the kernel would have to copy the 16 registers to the TCB.

Note

MSR and MRS instructions have visibility of both stacks.

5.4.1 Stacks

The stack model is independent of privileged mode, that is, Thread mode can use the process or main stack and be in user or privileged mode. All four combinations of stack and privilege are possible. For a basic protected thread model, the user threads run in Thread mode using the process stack, and the kernel and the interrupts run privileged using the main stack.

Note

Privilege alone does not prevent corruption of stacks, whether malicious or accidental. A memory protection scheme of one form or another is required to isolate the user code. That is, you must prevent the user code from writing to memory it does not own, including other stacks.

5.4.2 Privilege

Privilege controls access rights, and is decoupled from all other concepts in ARMv7-M. Code can be privileged, with full access rights, or unprivileged, with limited access rights. Access rights affect ability to:

- Use or not use certain instructions such as MSR fields.
- Access *System Control Space* (SCS) registers.
- Access memory or peripherals, based on system design. The processor indicates to the system whether the code making an access is privileged and so the system can enforce restrictions on non-privileged access.
- Access rules to memory locations based on an MPU. When fitted with an MPU, the access restrictions can control what memory can be read, written, and executed.

Only Thread mode can be unprivileged. All exceptions are privileged.

5.5 Pre-emption

The following sections describe the behavior of the processor when it takes an exception:

- *Stacking*
- *Late-arriving* on page 5-15
- *Tail-chaining* on page 5-14.

5.5.1 Stacking

When the processor invokes an exception, it automatically stores the following eight registers to the SP in the following order:

- *Program Counter (PC)*
- *Processor Status Register (xPSR)*
- r0-r3
- r12
- *Link Register (LR).*

The SP is decremented by eight words by the completion of the stack push. Figure 5-1 shows the contents of the stack after an exception pre-empts the current program flow.

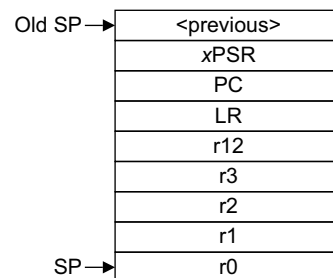


Figure 5-1 Stack contents after pre-emption

Note

- Figure 5-1 shows the order on the stack.
- If STKALIGN is set in the Configuration Control Register then an extra word can be inserted before the stacking takes place. See *Configuration Control Register* on page 8-26.

After returning from the ISR, the processor automatically pops the eight registers from the stack. Interrupt return is passed as a data field in the LR, so ISR functions can be normal C/C++ functions, and do not require a veneer.

Table 5-4 describes the steps that the processor takes before it enters an ISR.

Table 5-4 Exception entry steps

Action	Restartable?	Description
Push eight registers ^a	No.	Pushes xPSR, PC, r0, r1, r2, r3, r12, and LR on selected stack.
Read vector table	Yes. Late-arriving exception can cause restart.	Reads vector table from memory based on table base + (exception number 4). Read on the ICode bus can be done simultaneously with register pushes on the DCode bus.
Read SP from vector table	No.	On Reset only, updates SP to top of stack from vector table. Other exceptions do not modify SP except to select stack, push, and pop.
Update PC	No.	Updates PC with vector table read location. Late-arriving exceptions cannot be processed until the first instruction starts to execute.
Load pipeline	Yes. Pre-emption reloads pipeline from new vector table read.	Loads instructions from location pointed to by vector table. This is done in parallel with register push.
Update LR	No.	LR is set to EXC_RETURN to exit from exception. EXC_RETURN is one of 16 values as defined in <i>ARMv7-M Architecture Reference Manual</i> .

a. When tail-chaining, this step is skipped.

Figure 5-2 on page 5-13 shows an example of exception entry timing.

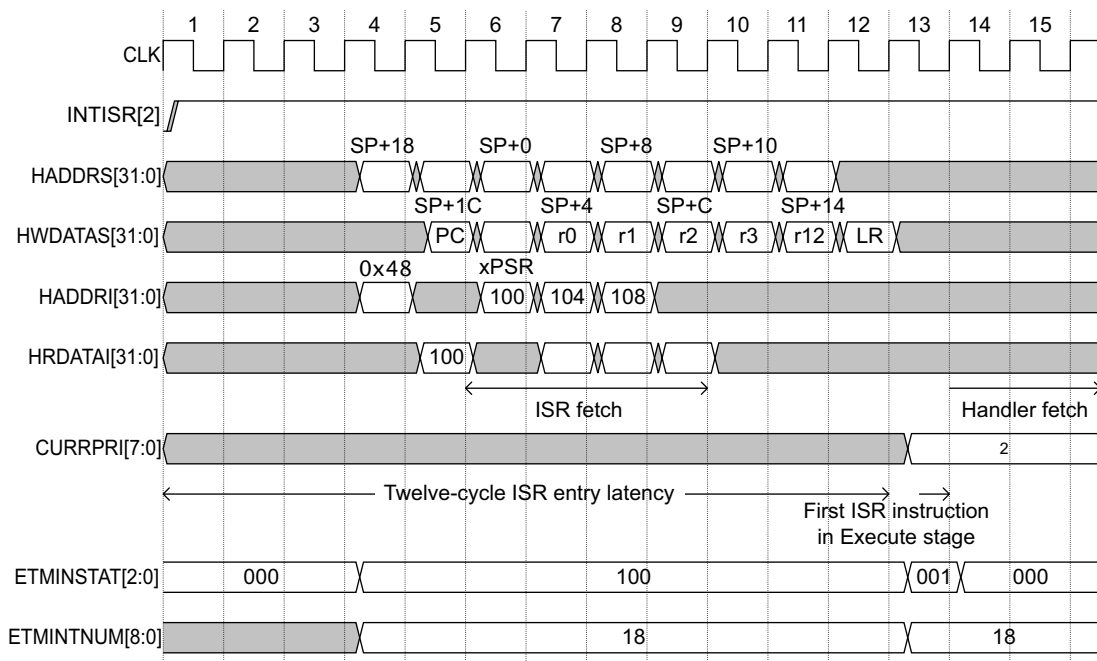


Figure 5-2 Exception entry timing

The NVIC indicates to the processor core, in the cycle after **INTISR[2]** was received, that an interrupt has been received, and the processor initiates the stack push and vector fetch in the following cycle.

When the stack push has completed, the first instruction of the ISR enters the execute stage of the pipeline. In the cycle that the ISR enters execute:

- **ETMINSTAT[2:0]** indicates that the ISR has been entered (3'b001). This is a 1-cycle pulse.
- **CURRPRI[7:0]** indicates the priority of the active interrupt. **CURRPRI** remains asserted throughout the duration of the ISR. **CURRPRI** becomes valid when **ETMINTSTAT** indicates that the ISR has been entered (3'b001).
- **ETMINTNUM[8:0]** indicates the number of the active interrupt. **ETMINTNUM** remains asserted throughout the duration of the ISR. **ETMINTNUM** becomes valid when **ETMINTSTAT** indicates that the ISR has been entered (3'b001). Prior to that it indicates which ISR is being fetched.

Figure 5-2 shows that there is a 12-cycle latency from asserting the interrupt to the first instruction of the ISR executing.

5.6 Tail-chaining

Tail-chaining is back-to-back processing of exceptions without the overhead of state saving and restoration between interrupts. The processor skips the pop of eight registers and push of eight registers when exiting one ISR and entering another because this has no effect on the stack contents.

The processor tail-chains if a pending interrupt has higher priority than all stacked exceptions.

Figure 5-3 shows an example of tail-chaining. If a pending interrupt has higher priority than the highest-priority stacked exception, the stack push or pop is omitted, and the processor immediately fetches the vector for the pending interrupt. The ISR that is tail-chained into starts execution six cycles after exiting the previous ISR.

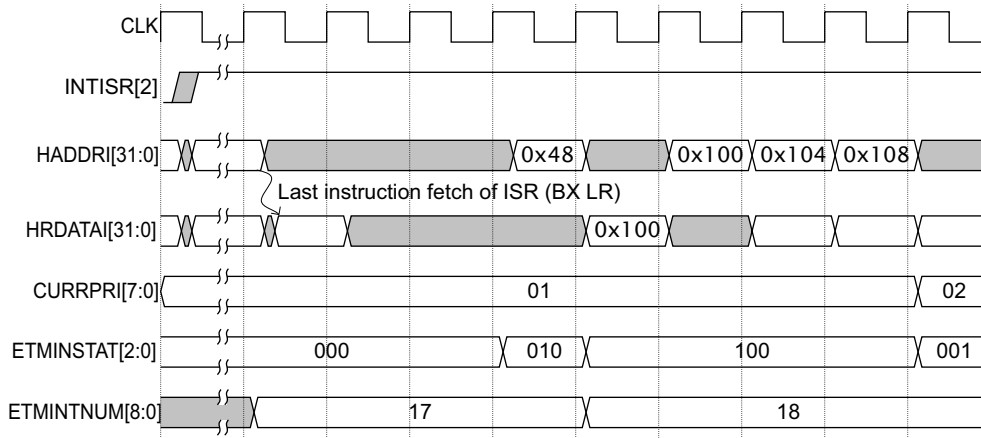


Figure 5-3 Tail-chaining timing

On return from the last ISR, **INTISR[2]** is of higher priority than any stacked ISR, or other pended interrupt, and so the processor tail-chains to the ISR corresponding to **INTISR[2]**. In the cycle that the ISR for **INTISR[2]** enters execute:

- **ETMINSTAT[2:0]** indicates that the ISR has been entered (3'b001). This is a 1-cycle pulse.
- **CURRPRI[7:0]** indicates the priority of the active interrupt. **CURRPRI** remains asserted throughout the duration of the ISR.
- **ETMINTNUM[8:0]** indicates the number of the active interrupt. **ETMINTNUM** remains asserted throughout the duration of the ISR.

Figure 5-3 shows that there is a 6-cycle latency when returning from the last ISR to executing the new ISR.

5.7 Late-arriving

A late-arriving interrupt can pre-empt a previous interrupt if the first instruction of the previous ISR has not entered the Execute stage, and the late-arriving interrupt has a higher priority than the previous interrupt.

A late-arriving interrupt causes a new vector address fetch and ISR prefetch. State saving is not performed for the late-arriving interrupt because it has already been performed for the initial interrupt and so does not have to be repeated.

Figure 5-4 shows an example of late-arriving interrupts.

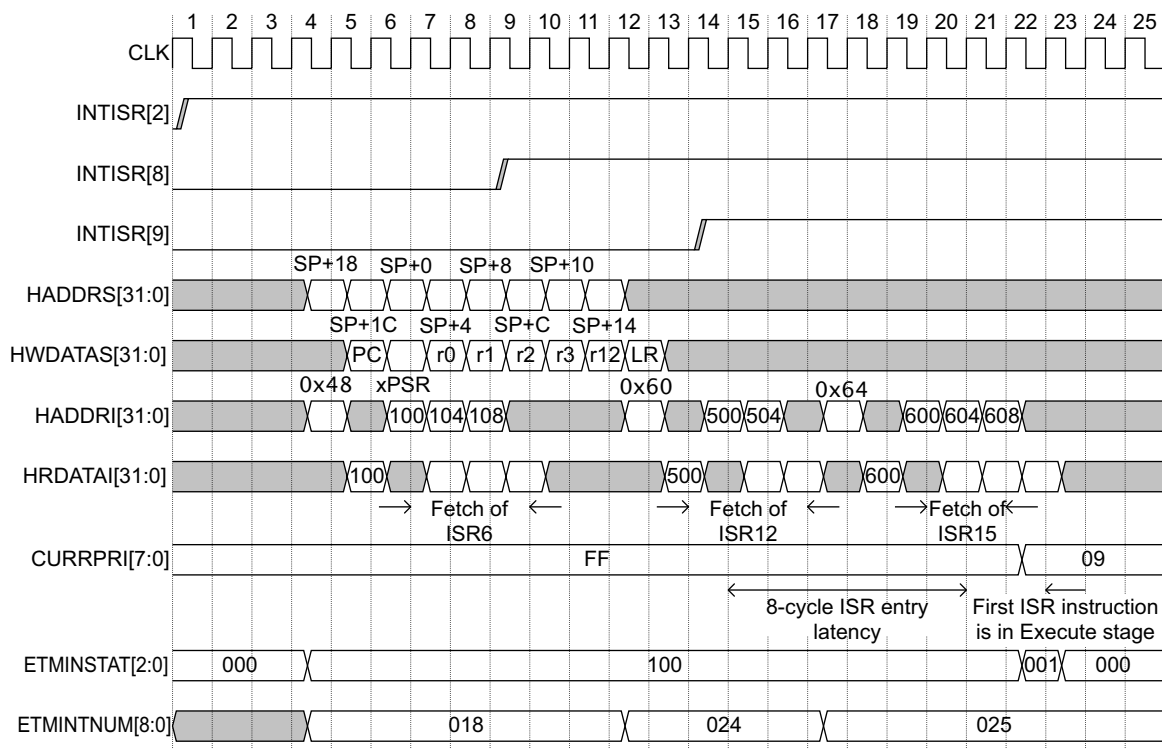


Figure 5-4 Late-arriving exception timing

In Figure 5-4, **INTISR[8]** pre-empt **INTISR[2]**. The state saving for **INTISR[2]** is already done and is not required to be repeated. Figure 5-4 shows the latest point at which **INTISR[8]** can pre-empt before the first instruction of the ISR for **INTISR[2]** enters Execute stage. A higher priority interrupt after that point is managed as a pre-emption.

Figure 5-4 on page 5-15 shows the latest point at which **INTISR[9]** can pre-empt before the first instruction of the ISR for **INTISR[8]** enters Fetch stage. The ISR fetch for **INTISR[8]** is aborted when **INTISR[9]** is received, and the processor then initiates the vector fetch for **INTISR[9]**. A higher priority interrupt after that point is managed as pre-emption.

In the cycle that the ISR for **INTISR[9]** enters execute:

- **ETMINSTAT[2:0]** indicates that the ISR has been entered (3'b001). This is a 1-cycle pulse.
- **CURRPRI[7:0]** indicates the priority of the active interrupt. **CURRPRI** remains asserted throughout the duration of the ISR.
- **ETMINTNUM[8:0]** indicates the number of the active interrupt. **ETMINTNUM** remains asserted throughout the duration of the ISR.

5.8 Exit

The last instruction of an ISR loads the PC with value 0xFFFFFFF that was LR on exception entry. This indicates to the processor that the ISR is complete, and the processor initiates the exception exit sequence. See *Returning the processor from an ISR* on page 5-18 for the instructions that you can use to return the processor from an ISR.

5.8.1 Exception exit

When returning from an exception, the processor is either:

- tail-chaining to a pending exception if the pending exception is of higher priority than all stacked exceptions
- returning to the last stacked ISR if there are no pending exceptions or if the highest priority stacked exception is of higher priority than the highest priority pending exception
- returning to the Thread mode if there are no pending or stacked exceptions.

Table 5-5 describes the postamble sequence.

Table 5-5 Exception exit steps

Action	Description
Pop eight registers	Pops PC, xPSR, r0, r1, r2, r3, r12 and LR from stack selected by EXC_RETURN and adjusts SP, if not pre-empted.
Load current active interrupt number ^a and reverse stack-alignment adjustment	Loads current active interrupt number from bits [8:0] of stacked IPSR word. The processor uses this to track which exception to return to and to clear the activation bit on return. When bits [8:0] are zero, the processor returns to Thread Mode.
Select SP	If returning to an exception, SP is SP_main. If returning to Thread Mode, SP can be SP_main or SP_process.

a. Because of dynamic priority changes, the NVIC uses interrupt numbers instead of interrupt priorities to determine which ISR is current.

Figure 5-5 on page 5-18 shows an example of exception exit timing.

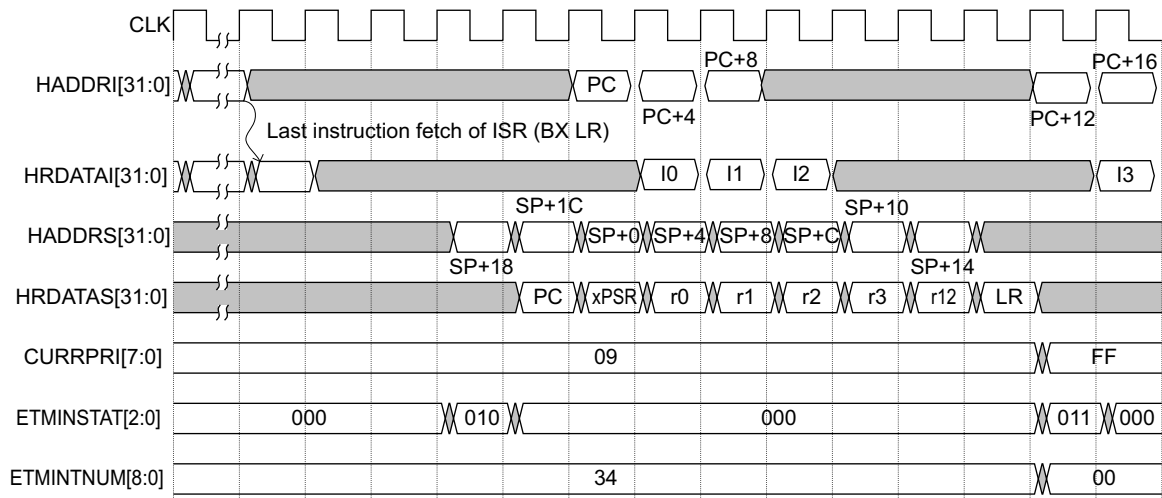


Figure 5-5 Exception exit timing

ETMINSTAT indicates:

- 3'b010 to show that the ISR has exited. **ETMINTNUM** shows the number of the ISR that exited.
- 3'b011 in the cycle after interrupt exit if a previous stacked ISR is being returned to. **ETMINTNUM** shows the number of the interrupt that is being returned to.

———— **Note** ————

If a higher priority exception occurs during the stack pop, the processor abandons the stack pop, rewinds the stack pointer, and services the exception as a tail-chain case.

5.8.2 Returning the processor from an ISR

Exception returns occur when one of the following instructions loads a value of 0xFFFFFFFF into the PC:

- POP/LDM which includes loading the PC
- LDR with PC as a destination
- BX with any register.

When used in this way, the value written to the PC is intercepted and is referred to as the **EXC_RETURN** value. EXC_RETURN[3:0] provides return information as defined in Table 5-6.

Table 5-6 Exception return behavior

EXC_RETURN[3:0]	Description
0bXXX0	Reserved.
0b0001	Return to Handler mode. Exception return gets state from the main stack. On return execution uses the main stack.
0b0011	Reserved.
0b01X1	Reserved.
0b1001	Return to Thread mode. Exception return gets state from the main stack. On return execution uses the main stack.
0b1101	Return to Thread mode. Exception return gets state from the process stack. On return execution uses the process stack.
0b1X11	Reserved.

Reserved entries in this table result in a chained exception to a Usage Fault.

If an EXC_RETURN value is loaded into the PC when in Thread mode, or from the vector table, or by any other instruction, the value is treated as an address, not as a special value. This address range is defined to have *Execute Never* (XN) permissions, and results in a MemManage fault.

5.9 Resets

The NVIC is reset at the same time as the core and controls the release of reset into the core. As a result, the behavior of reset is predictable. Table 5-7 shows the reset behavior.

Table 5-7 Reset actions

Action	Description
NVIC resets, holds core in reset	NVIC clears most of its registers. The processor is in Thread mode, priority is privileged, and the stack is set to Main.
NVIC releases core from reset	NVIC releases core from reset.
Core sets stack	Core reads the start SP, SP_main, from vector-table offset 0.
Core sets PC and LR	Core reads the start PC from vector-table offset. LR is set to 0xFFFFFFFF.
Reset routine runs	NVIC has interrupts disabled, and NMI and Hard Fault are not disabled.

For more information about resets, see Chapter 6 *Clocking and Resets*.

5.9.1 Vector Table and Reset

The vector table at location 0 provides the vector table at reset. It must contain at least four values:

- stack top address
- reset routine location
- NMI ISR location
- Hard Fault ISR location.

When interrupts are enabled, the vector table regardless of location, points to all mask-enabled exceptions. Also, the SVCall ISR location is populated if the SVC instruction is used.

An example of a full vector table:

```
unsigned int stack_base[STACK_SIZE];
void ResetISR(void);
void NmiISR(void);
...
ISR_VECTOR_TABLE vector_table_at_0
{
    stack_base + sizeof(stack_base),
    ResetISR,
    NmiISR,
    FaultISR,
```

```

0,          // Populate if using MemManage (MPU)
0,          // Populate if using Bus fault
0,          // Populate if using Usage Fault
0, 0, 0, 0, // reserved slots
SVCallISR,
0,          // Populate if using a debug monitor
0,          // Reserved
0,          // Populate if using pendable service request
0,          // Populate if using SysTick
// external interrupts start here
Timer1ISR,
GpioInISR
GpioOutISR,
I2CIsr
};

```

Note

Vector table entries are ARM/Thumb interworking compatible. This causes bit [0] of the vector value to load into the EPSR T-bit on exception entry. Creating a table entry with bit [0] clear generates an INVSTATE fault on the first instruction of the handler corresponding to this vector.

5.9.2 Intended boot-up sequence

A normal reset routine follows the steps shown in Table 5-8. A C/C++ runtime can perform the first three steps and then call `main()`.

Table 5-8 Reset boot-up behavior

Action	Description
Initialize variables	Any global/static variables must be setup. This includes initializing the BSS variable to 0, and copying initial values from ROM to RAM for non-constant variables.
[Setup stacks]	If more than one stack is be used, the other banked SPs must be initialized. The current SP can also be changed to Process from Main.
Initialize any runtime	Optionally make calls to C/C++ runtime init code to enable use of heap, floating point, or other features. This is normally done by <code>__main</code> from the C/C++ library.
[Initialize any peripherals]	Setup peripherals before interrupts are enabled. This can call to setup each peripheral to be used in the application.
[Switch ISR vector table]	Optionally change vector table from Code area, @0, to a location in SRAM. This is only done to optimize performance or enable dynamic changes.
[Setup Configurable Faults]	Enable Configurable faults and set their priorities.

Table 5-8 Reset boot-up behavior (continued)

Action	Description
Setup interrupts	Setup priority levels and masks.
Enable interrupts	Enable interrupts. Enable the interrupt processing in the NVIC. It is not desirable to have these occur while they are being enabled. If there are more than 32 interrupts, it takes more than one Set-Enable Register. PRIMASK can be used through CPS or MSR to mask interrupts until ready.
[Change Privilege]	[Change Privilege]. The Thread mode privilege can be changed to user if required. This must normally be handled by invoking the SVCcall handler.
Loop	If sleep-on-exit is enabled, control never returns after the first interrupt/exception is taken. If sleep-on-exit is selectively enabled/disabled, this loop can manage cleanup and executive tasks. If sleep-on-exit is not used, the loop is free and can use WFI (sleep-now) when required.

———— **Note** ————

Entries in Table 5-8 on page 5-21 that are bracketed are optional actions.

Example of reset routine

The reset routine is responsible for starting up the application and then enabling interrupts. There are three methods for involving the reset ISR after interrupt processing is performed. This is called the main loop part of the Reset ISR and the three examples are shown in Example 5-1, Example 5-2 on page 5-23, and Example 5-3 on page 5-23.

Example 5-1 Reset routine with pure sleep on exit (Reset routine does no main loop work)

```
void reset()
{
    // do setup work (initialize variables, initialize runtime if wanted,
    setup peripherals, etc)
    nvic[INT_ENA] = 1;    // enable interrupts
    nvic_regs[NV_SLEEP] |= NV_SLEEP_ON_EXIT; // will not normally come back after
                                           1st exception

    while (1)
        wfi();
}
```

Example 5-2 Reset routine with selected Sleep model using WFI

```

void reset()
{
    extern volatile unsigned exc_req;
    // do setup work (initialize variables, initialize runtime if wanted,
    // setup peripherals, etc)
    nvic[INT_ENA] = 1;    // enable interrupts
    while (1)
    {
        // do some work for (exc_req = FALSE; exc_req == FALSE; )
        wfi(); // sleep now - wait for interrupt
        // do some post exception checking/cleanup
    }
}

```

Example 5-3 Reset routine with selected Sleep on exit cancelled by ISRs that require attention

```

void reset()
{
    // do setup work (initialize variables, initialize runtime if wanted,
    // setup peripherals, etc)
    nvic[INT_ENA] = 1;    // enable interrupts
    while (1)
    {
        // We are slept until an exception clears sleep on exit state so that we
        // can post-process/cleanup.
        nvic_regs[NV_SLEEP] |= NV_SLEEP_ON_EXIT;
        while (nvic_regs[NV_SLEEP] & NV_SLEEP_ON_EXIT)
            wfi(); // sleep now - wait for interrupt which clears
        // do some post exception checking/cleanup
    }
}

```

Note

An executive is not required in the Reset routine because an ISR activation can enact priority level changes. This ensures faster response to changing loads, and uses priority boosting, to solve priority inversions, to ensure fine grain support. Thread mode is used for the user code for *Real Time Operating System* (RTOS) models using threads and privilege.

5.10 Exception control transfer

The processor transfers control to an ISR following the rules shown in Table 5-9.

Table 5-9 Transferring to exception processing

Processor activity at assertion of exception	Transfer to exception processing
Non-memory instruction	Takes exception on completion of cycle, before the next instruction.
Load/store single	Completes or abandons depending on bus status. Takes exception on the next cycle, depending on the bus wait states.
Load/store multiple	Completes or abandons current register and sets continuation counter into EPSR. Takes exception on the next cycle, depending on bus permission and <i>Interruptible-Continuable Instruction</i> (ICI) rules. For more information on ICI rules, see the <i>ARMv7-M Architecture Reference Manual</i> .
Exception entry	This is a late-arriving exception. If it has higher priority than the exception being entered, then the processor cancels the exception entry actions and takes the late-arriving exception. Late arriving results in a decision change (vector table) at interrupt processing time. When you enter a new handler, that is the first ISR instruction, normal pre-emption rules apply, and it is no longer classed as a late-arrival.
Tail-chaining	This is a late-arriving exception. If it has higher priority than the one being tail-chained, the processor cancels the preamble and takes the late-arriving exception.
Exception postamble	If the new exception has higher priority than the stacked exception to which the processor is returning, the processor tail-chains the new exception.

5.11 Setting up multiple stacks

To implement multiple stacks, the application must take these actions:

- use the MSR instruction to set up the Process_SP register
- if using an MPU, protect the stacks appropriately
- initialize the stack and privilege of the Thread mode.

If the privilege of Thread mode is changed from privileged to user, only another ISR, such as SVCcall, can change the privilege back from user to privileged.

The stack in Thread mode can be changed from main to process or from process to main, but doing so affects its access to the local variables of the thread. It is better to have another ISR change the stack used in Thread mode. The following shows an example boot sequence:

1. Call setup routine to:
 - a. Set up other stacks using MSR.
 - b. Enable the MPU to support base regions, if any.
 - c. Invoke all boot routines.
 - d. Return from setup routine.
2. Change Thread mode to unprivileged.
3. Use SVC to invoke the kernel. Then the kernel:
 - a. Starts threads.
 - b. Uses MRS to read the SP for the current user thread and save it in its TCB.
 - c. Uses MSR to set the SP for the next thread. This is usually SP_process.
 - d. Sets up the MPU for the newly current thread, if necessary.
 - e. Returns into the newly current thread.

Example 5-4 shows a modification to the EXC_RETURN value in the ISR to return using PSP.

Example 5-4 Modification to the EXC_RETURN value in the ISR

```

; First time use of PSP, run from a Handler with RETTBASE == 1
LDR r0, PSPValue      ; acquire value for new Process stack
MSR PSP, r0           ; set Process stack value
ORR lr, lr, #4        ; change EXC_RETURN for return on PSP
BX lr                 ; return from Handler to Thread

```

Example 5-5 on page 5-26 shows how to implement a simple context switcher after the switch to Thread on PSP.

Example 5-5 Implement a simple context switcher

```

; Example Context Switch (Assumes Thread is already on PSP)
MRS r12, PSP                ; Recover PSP into R12
STMDB r12!, {r4-r11, LR}    ; Push non-stack registers
LDR r0, =OldPSPValue        ; Get pointer to old Thread Control Block
STR r12, [r0]               ; Store SP into Thread Control Block
LDR r0, =NewPSPValue        ; Get pointer to new Thread Control Block
LDR r12, [r0]               ; Acquire new Process SP
LDMIA r12!, {r4-r11, LR}    ; Restore non-stacked registers
MSR PSP, r12                ; Set PSP to R12
BX lr                      ; Return back to Thread

```

Note

In Example 5-4 on page 5-25 and Example 5-5, the only time the decision to move Thread from MSP to PSP can be made, or the non-stacked registers can be guaranteed not to have been modified by a stacked Handler, is when there is only one active ISR/Handler.

5.12 Abort model

Four events can generate a fault:

- An instruction fetch or vector table load bus error.
- A data access bus error.
- Internally-detected error such as an undefined instruction or an attempt to change state with a BX instruction. Fault status registers in the NVIC indicate the causes of the faults.
- MPU fault because of privilege violation or unmanaged region.

There are two kinds of fault handler:

- the fixed-priority Hard Fault
- the settable-priority local faults.

5.12.1 Hard Fault

Only Reset and NMI can pre-empt the fixed priority Hard Fault. A Hard Fault can pre-empt any other exception other than Reset NMI or another Hard Fault.

Note

Code that uses FAULTMASK acts as a Hard Fault and so follows the same rules as a Hard Fault.

Secondary bus faults do not escalate because a pre-empting fault of the same type cannot pre-empt itself. This means that if a corrupted stack causes a fault, the fault handler still executes even though the stack pushes for the handler failed. The fault handler can operate, but the stack contents are corrupted.

5.12.2 Local faults and escalation

Local faults are categorized according to their cause. See Table 5-10 on page 5-28. When enabled, local fault handlers process all normal faults. However, a local fault escalates to a Hard Fault when:

- A local fault handler causes the same kind of fault as the one it is servicing.
- A local fault handler causes a fault with the same or higher priority.
- An exception handler causes a fault with the same or higher priority.
- The local fault is not enabled.

Table 5-10 lists the local faults.

Table 5-10 Faults

Fault	Bit name	Handler	Notes	Trap enable bit
Reset	Reset cause	Reset	Any form of reset.	RESETVCATCH
Vector Read error	VECTTBL	HardFault	Bus error returned when reading the vector table entry.	INTERR
uCode stack push error	STKERR	BusFault	Failure when saving context using hardware - bus error returned.	INTERR
uCode stack push error	MSTKERR	MemManage	Failure when saving context using hardware - MPU access violation.	INTERR
uCode stack pop error	UNSTKERR	BusFault	Failure when restoring context using hardware - bus error returned.	INTERR
uCode stack pop error	MUNSKERR	MemManage	Failure when restoring context using hardware - MPU access violation.	INTERR
Escalated to Hard Fault	FORCED	HardFault	Fault occurred and handler is equal or higher priority than current, including fault within fault when priority does not enable, or Configurable fault disabled. Includes SVC, BKPT and other kinds of faults.	HARDERR
MPU mismatch	DACCVIOL	MemManage	Violation or fault on MPU as a result of data access.	MMERR
MPU mismatch	IACCVIOL	MemManage	Violation or fault on MPU as a result of instruction address.	MMERR
Pre-fetch error	IBUSERR	BusFault	Bus error returned because of instruction fetch. Faults only if makes it to execute. Branch shadow can fault and be ignored.	BUSERR
Precise data bus error	PRECISERR	BusFault	Bus error returned because of data access, and was precise, points to instruction.	BUSERR

Table 5-10 Faults (continued)

Fault	Bit name	Handler	Notes	Trap enable bit
Imprecise data bus error	IMPRECISERR	BusFault	Late bus error because of data access. Exact instruction is no longer known. This is pended and not synchronous. It does not cause FORCED.	BUSERR
No Coprocessor	NOCP	UsageFault	Truly does not exist, or not present bit.	NOCPELL
Undefined Instruction	UNDEFINSTR	UsageFault	Unknown instruction.	STATERR
Attempt to execute an instruction when in an invalid ISA state. For example, not Thumb	INVSTATE	UsageFault	Attempt to execute in an invalid EPSR state. For example, after a BX type instruction has changed state. This includes states after return from exception including inter-working states.	STATERR
Return to PC=EXC_RETURN when not enabled or with invalid magic number	INVPC	UsageFault	Illegal exit, caused either by an illegal EXC_RETURN value, an EXC_RETURN and stacked EPSR value mismatch, or an exit while the current EPSR is not contained in the list of currently active exceptions.	STATERR
Illegal unaligned load or store	UNALIGNED	UsageFault	This occurs when any load-store multiple instruction attempts to access a non-word aligned location. It can be enabled to occur for any load-store that is unaligned to its size using the UNALIGN_TRP bit.	CHKERR
Divide By 0	DIVBYZERO	UsageFault	This can be enabled to occur when SDIV or UDIV is executed with a divisor of 0, and the DIV_0_TRP bit is set.	CHKERR
SVC	-	SVCall	System request (Service Call).	-

Table 5-11 shows debug faults.

Table 5-11 Debug faults

Fault	Flag	Notes	Trap enable bit
Internal halt request	HALTED	NVIC request from, for example, step, core halt	-
Breakpoint	BKPT	SW breakpoint from patched instruction or FPB	-
Watchpoint	DWTTRAP	Watchpoint match in DWT	-
External	EXTERNAL	EDBGRQ line asserted	-
Vector catch	VCATCH	Vector catch triggered. Corresponding FSR contains the primary cause of the exception.	VC_xxx bit(s) or RESETVCATCH set

5.12.3 Fault status registers and fault address registers

Each fault has a fault status register with a flag for that fault.

There are:

- three configurable fault status registers that correspond to the three configurable fault handlers
- one hard fault status register
- one debug fault status register.

Depending on the cause, one of the five status registers has a bit set.

There are two *Fault Address Registers* (FAR):

- *Bus Fault Address Register* (BFAR)
- *Memory Fault Address Register* (MFAR).

A flag in the corresponding fault status register indicates when the address in the fault address register is valid.

———— **Note** ————

BFAR and MFAR are the same physical register. Because of this, the BFARVALID and MFARVALID bits are mutually exclusive.

Table 5-12 on page 5-31 shows the fault status registers and two fault address registers.

Table 5-12 Fault status and fault address registers

Status Register name	Handler	Address Register name	Description
HFSR	Hard Fault	-	Escalation and Special
MMSR	Mem Manage	MMAR	MPU faults
BFSR	Bus Fault	BFAR	Bus faults
UFSR	Usage Fault	-	Usage fault
DFSR	Debug Monitor or Halt	-	Debug traps

5.13 Activation levels

When no exceptions are active, the processor is in Thread mode. When an ISR or fault handler is active, the processor enters Handler mode. Table 5-13 lists the privilege and stacks of the activation levels.

Table 5-13 Privilege and stack of different activation levels

Active exception	Activation level	Privilege	Stack
None	Thread mode	Privileged or user	Main or process
ISR active	Asynchronous pre-emption level	Privileged	Main
Fault handler active	Synchronous pre-emption level	Privileged	Main
Reset	Thread mode	Privileged	Main

Table 5-14 summarizes the transition rules for all exception types and how they relate to the access rules and stack model.

Table 5-14 Exception transitions

Active Exception	Triggering event	Transition type	Privilege	Stack
Reset	Reset signal	Thread	Privileged or user	Main or process
ISR ^a or NMI ^b	Set-pending software instruction or hardware signal	Asynchronous pre-emption	Privileged	Main
Fault:				
Hard fault	Escalation	Synchronous pre-emption	Privileged	Main
Bus fault	Memory access error			
No CP ^c fault	Absent CP access			
Undefined instruction fault	Undefined instruction			
Debug monitor	Debug event when halting not enabled	Synchronous	Privileged	Main
SVC ^d	SVC instruction			
External interrupt				

a. Interrupt service routine.
b. Nonmaskable interrupt.
c. Coprocessor.
d. Software interrupt.

Table 5-15 shows exception subtype transitions.

Table 5-15 Exception subtype transitions

Intended activation subtype	Triggering event	Activation	Priority effect
Thread	Reset signal	Asynchronous	Immediate, thread is lowest
ISR/NMI	HW signal or set-pend	Asynchronous	Pre-empt or tail-chain according to priority
Monitor	Debug event ^a	Synchronous	If priority less than or equal to current, hard fault
SVCcall	SVC instruction	Synchronous	If priority less than or equal to current, hard fault
PendSV	Software pend request	Chain	Pre-empt or tail-chain according to priority
UsageFault	Undefined instruction	Synchronous	If priority greater than or equal to current, hard fault
NoCpFault	Access to absent CP	Synchronous	If priority greater than or equal to current, hard fault
BusFault	Memory access error	Synchronous	If priority greater than or equal to current, hard fault
MemManage	MPU mismatch	Synchronous	If priority greater than or equal to current, hard fault
HardFault	Escalation	Synchronous	Higher than all except NMI
FaultEscalate	Escalate request from Configurable fault handler	Chain	Boosts priority of local handler to same as hard fault so it can return and chain to Configurable Fault handler

a. While halting not enabled.

5.14 Flowcharts

This section summarizes interrupt flow with:

- *Interrupt handling*
- *Pre-emption* on page 5-35
- *Return* on page 5-35.

5.14.1 Interrupt handling

Figure 5-6 shows how instructions execute until pre-empted by a higher-priority interrupt.

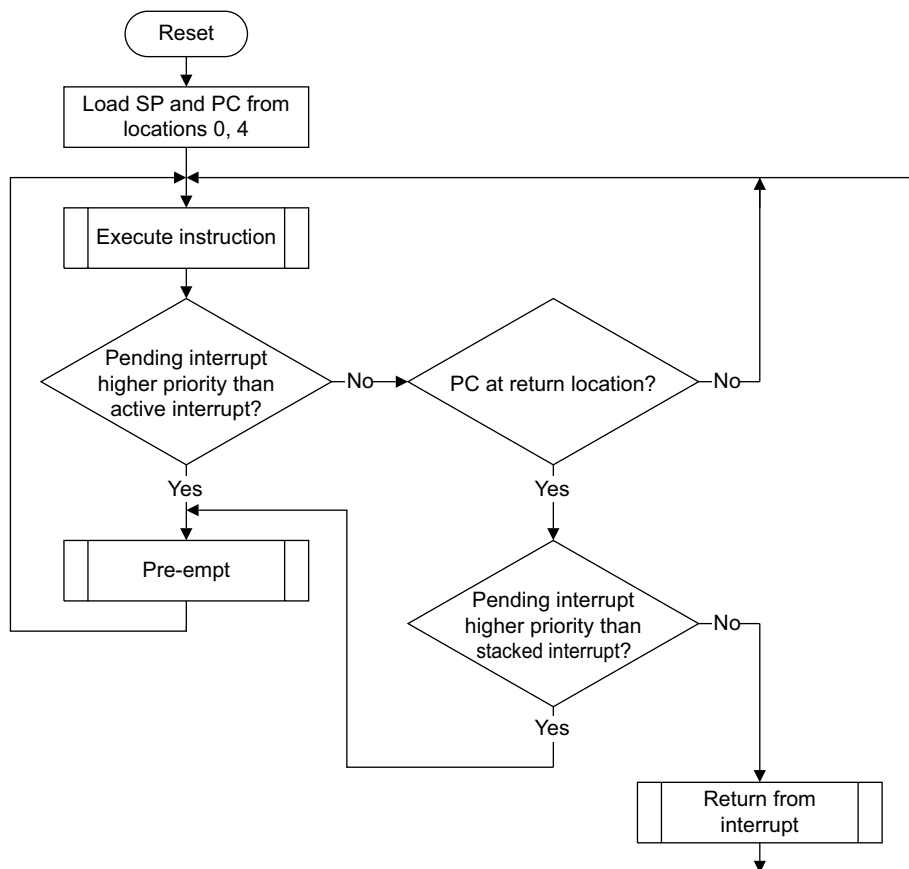


Figure 5-6 Interrupt handling flowchart

5.14.2 Pre-emption

Figure 5-7 shows what happens when an exception pre-empts the current ISR.

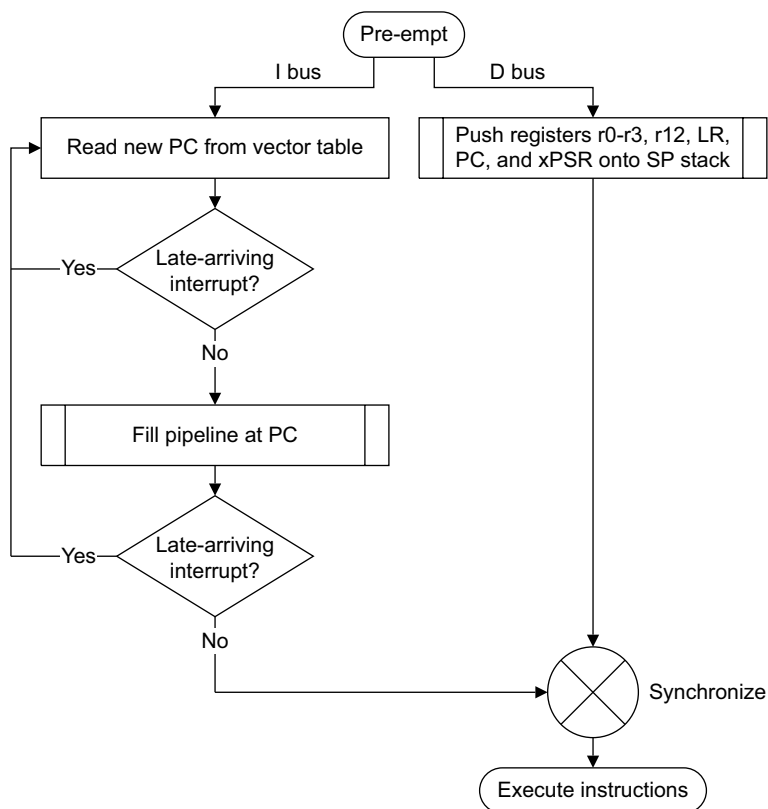
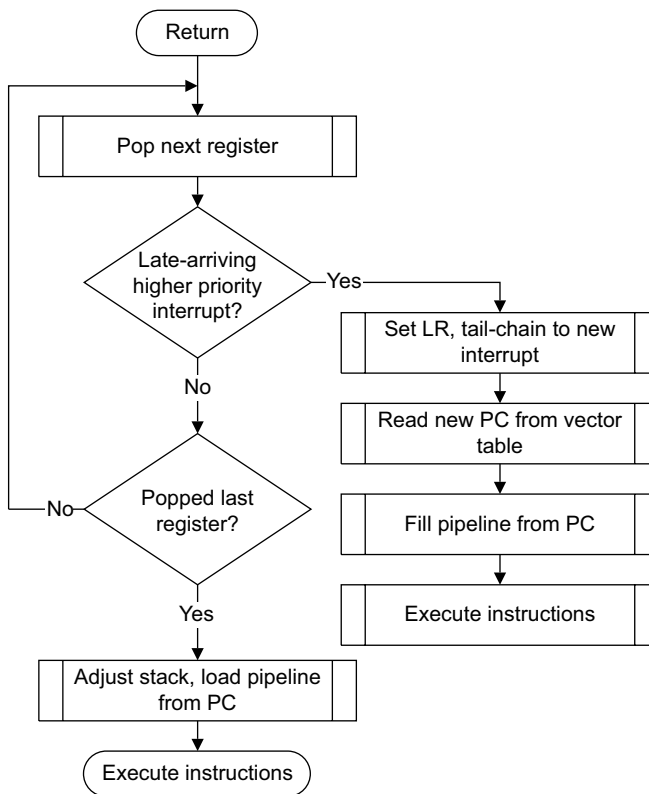


Figure 5-7 Pre-emption flowchart

5.14.3 Return

Figure 5-8 on page 5-36 shows how the processor restores the stacked ISR or tail-chains to a late-arriving interrupt with higher priority than the stacked ISR.

**Figure 5-8 Return from interrupt flowchart**

Chapter 6

Clocking and Resets

This chapter describes the processor clocking and resets. It contains the following sections:

- *Clocking* on page 6-2
- *Resets* on page 6-4
- *Cortex-M3 reset modes* on page 6-5.

6.1 Clocking

The processor has three functional clock inputs. Table 6-1 describes the processor clocks.

Table 6-1 Cortex-M3 processor clocks

Clock	Domain	Description
FCLK	Processor	Free running processor clock, used for sampling interrupts and clocking debug blocks. FCLK ensures that interrupts can be sampled, and sleep events can be traced, while the processor is sleeping.
HCLK	Processor	Processor clock.
DAPCLK	Processor	Debug port <i>Advanced High-performance Bus Access Port</i> (AHB-AP) clock.

FCLK and **HCLK** are synchronous to each other. **FCLK** is a free running version of **HCLK**, and therefore must always be the same frequency when not in sleep mode. For more information, see Chapter 7 *Power Management*. **FCLK** and **HCLK** must be balanced with respect to each other, with equal latencies into the processor.

The processor is integrated with components for debug and trace. Your macrocell might contain some, or all, of the clocks shown in Table 6-2.

Table 6-2 Cortex-M3 macrocell clocks

Clock	Domain	Description
TRACECLKIN	TPIU	Clocks the output of the TPIU
DBGCLK	SW-DP	Debug clock
SWCLKTCK	SWJ-DP	Debug clock

SWCLKTCK is the clock for the debug interface domain of the SWJ-DP. In JTAG mode this is equivalent to **TCK**. In Serial Wire Mode this is the Serial Wire clock. It is asynchronous to all other clocks. **DBGCLK** is the clock for the debug interface domain of SW-DP. It is asynchronous to the other clocks.

TRACECLKIN is the reference clock for the *Trace Port Interface Unit* (TPIU). It is asynchronous to the other clocks.

Note

SWCLKTCK, **DBGCLK**, and **TRACECLKIN** only require to be driven if your implementation contains *Serial Wire JTAG Debug Port* (SWJ-DP), *Serial Wire Debug Port* (SW-DP), and TPIU blocks respectively. Otherwise, the clock inputs must be tied off.

Note

The processor also contains a **STCLK** input. This port is not a clock. It is a reference input for the SysTick counter, and it must be less than half the frequency of **FCLK**. **STCLK** is synchronized internally by the processor to **FCLK**.

6.2 Resets

The processor has three reset inputs. Table 6-3 describes the reset inputs.

Table 6-3 Reset inputs

Reset input	Description
PORESETn	Resets the entire processor system with the exception of SWJ-DP
SYSRESETn	Resets the entire processor system with the exception of debug logic in the: <ul style="list-style-type: none">Nested Vectored Interrupt Controller (NVIC)Flash Patch and Breakpoint (FPB)Data Watchpoint and Trace (DWT)Instrumentation Trace Macrocell (ITM)AHB-AP.
nTRST	SWJ-DP reset

————— **Note** —————

nTRST resets **SWJ-DP**. If your implementation does not contain SWJ-DP, this reset must be tied off.

6.3 Cortex-M3 reset modes

The reset signals present in the processor design enable you to reset different parts of the design independently. Table 6-4 shows the reset signals, and the combinations and possible applications that you can use them in.

Table 6-4 Reset modes

Reset mode	SYSRESETn	nTRST	PORESETn	Application
Power-on reset	x	0	0	Reset at power up, full system reset. Cold reset.
System reset	0	x	1	Reset of processor core and system components, excluding debug.
SWJ-DP reset	1	0	1	Reset of SWJ-DP logic.

Note

PORESETn resets a superset of the **SYSRESETn** logic.

6.3.1 Power-on reset

Figure 6-1 on page 6-6 shows the reset signals for the macrocell.

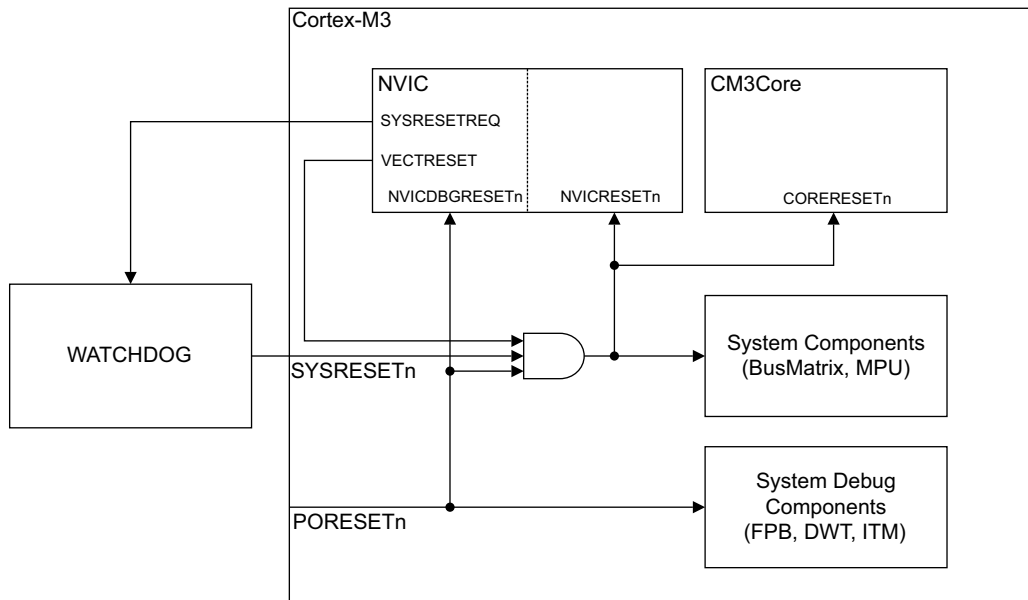


Figure 6-1 Reset signals

You must apply power-on or *cold* reset to the processor when power is first applied to the system. In the case of power-on reset, the falling edge of the reset signal, **PORESETn**, does not have to be synchronous to **HCLK**. Because **PORESETn** is synchronized within the processor, you do not have to synchronize this signal. Figure 6-2 shows the application of power-on reset. Figure 6-3 on page 6-7 shows the reset synchronizers within the processor.

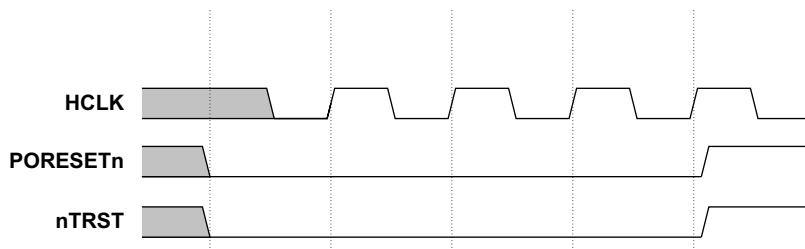


Figure 6-2 Power-on reset

It is recommended that you assert the reset signals for at least three **HCLK** cycles to ensure correct reset behavior. Figure 6-3 on page 6-7 shows the internal reset synchronization.

Note

You must consider LOCKUP from the Cortex-M3 system for inclusion in any external watchdog circuitry when an external debugger is not attached.

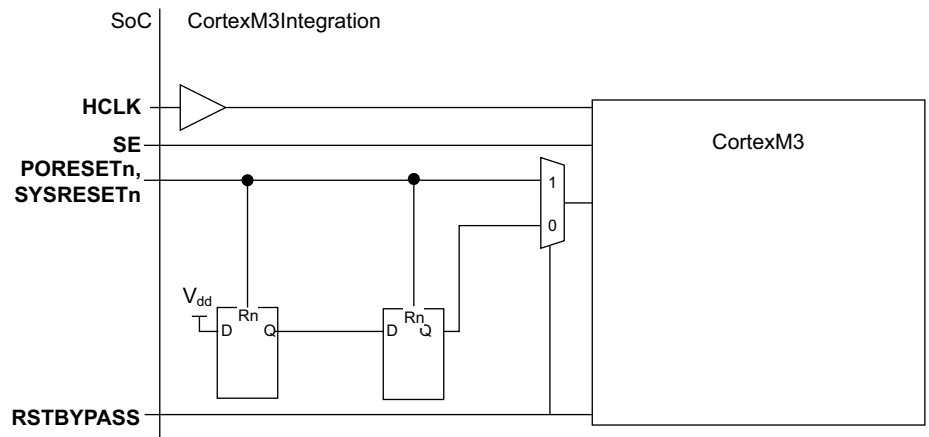


Figure 6-3 Internal reset synchronization

6.3.2 System reset

A system or *warm* reset initializes the majority of the macrocell, excluding the NVIC debug logic, FPB, DWT, and ITM. System reset typically resets a system that has been operating for some time, for example, watchdog reset.

SYSRESETn must be synchronized external to the processor. Figure 6-3 shows the example reset synchronization provided in CortexM3Integration.

Cortex-M3 exports a signal, **SYSRESETREQ**, that is asserted when the **SYSRESETREQ** bit of the Application Interrupt and Reset Control Register is set. For example, you can use this as an input to a watchdog timer as Figure 6-1 on page 6-6 shows.

6.3.3 SWJ-DP reset

nTRST reset initializes the state of the SWJ-DP controller. **nTRST** reset is typically used by the RealView™ ICE module for hot-plug connection of a debugger to a system.

nTRST enables initialization of the SWJ-DP controller without affecting the normal operation of the processor.

The **nTRST** signal must be asserted with regard to the **SWCLKTCK** clock because the SWJ-DP performs no synchronization.

6.3.4 SW-DP reset

SW-DP is reset with **DBGRESETn**. This reset must be synchronized to **DBGCLK**.

6.3.5 Normal operation

During normal operation, neither processor reset nor power-on reset is asserted. If the SWJ-DP port is not being used, the value of **nTRST** does not matter.

Chapter 7

Power Management

This chapter describes the processor power management functions. It contains the following sections:

- *About power management* on page 7-2
- *System power management* on page 7-3.

7.1 About power management

The processor extensively uses gated clocks to disable unused functionality, and disables inputs to unused functional blocks, so that only actively used logic consumes any dynamic power.

The ARMv7-M architecture supports system sleep modes that can stop the Cortex-M3 and system clocks for greater power reductions. These are described in *System power management* on page 7-3.

7.2 System power management

Writing to the System Control Register (see *System Control Register* on page 8-25) controls the Cortex-M3 system power states. Table 7-1 shows the supported sleep modes.

Table 7-1 Supported sleep modes

Sleep mechanism	Description
Sleep-now	The <i>Wait For Interrupt</i> (WFI) or the <i>Wait For Event</i> (WFE) instructions request the sleep-now model. These instructions cause the <i>Nested Vectored Interrupt Controller</i> (NVIC) to put the processor into the low-power state pending another exception. ^a
Sleep-on-exit	When the SLEEPONEXIT bit of the System Control Register is set, the processor enters the low-power state as soon as it exits the lowest priority ISR. The processor enters the low-power state without popping registers and a following exception is taken without having to push registers. The core stays in sleep state until another exception is pending. This is an automated WFI mode. <div style="text-align: center;">———— Note ————</div> Sleep-on-exit might return to base under various situations such as debug. Therefore, you must provide base code such as an idle loop or idle thread.
Deep-sleep	Deep-sleep is used in conjunction with Sleep-now and Sleep-on-exit. When the SLEEPDEEP bit of the System Control Register is set, the processor indicates to the system that deeper sleep is possible.

- a. The WFI instruction can complete even if no exception becomes active. Do not use it to detect the occurrence of an exception. WFI is normally used in an idle loop in the Thread mode. For more information on WFI, WFE, BASEPRI, and PRIMASK see the *ARMv7-M Architecture Reference Manual*.

The processor exports the following signals to indicate when the processor is sleeping:

SLEEPING This signal is asserted when in Sleep-now or Sleep-on-exit modes, and indicates that the clock to the processor can be stopped. On receipt of a new interrupt or event in the case of WFE, the NVIC de-asserts this signal, releasing the core from sleep. Halting the core also causes the sleep mode to be exited. *SLEEPING* on page 7-4 shows an example of **SLEEPING** usage.

SLEEPDEEP

This signal is asserted when in Sleep-now or Sleep-on-exit modes when the **SLEEPDEEP** bit of the System Control Register is set. This signal is routed to the clock manager and can gate the processor and system components including the *Phase Locked Loop* (PLL) to achieve greater

power savings. **SLEEPDEEP** is never asserted without **SLEEPING** also being asserted. *SLEEPDEEP* on page 7-5 shows an example of **SLEEPDEEP** usage.

7.2.1 SLEEPING

Figure 7-1 shows an example of how to reduce power consumption by gating the **HCLK** clock to the processor with **SLEEPING** in the low-power state. If necessary, you can also use **SLEEPING** to gate other system components. You can use the output signal **GATEHCLK** instead of creating your own clock gate enable term.

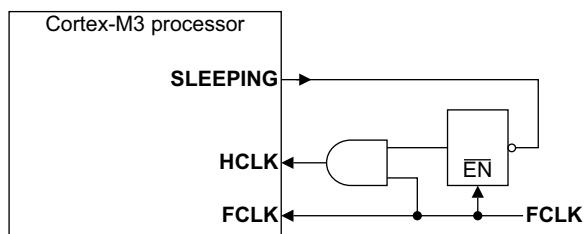


Figure 7-1 SLEEPING power control example

To detect interrupts, the processor must receive the free-running **FCLK** at all times, unless the WIC is in use. **FCLK** clocks:

- A small amount of logic in the NVIC that detects interrupts.
- The *Data Watchpoint and Trace (DWT)* and *Instrumentation Trace Macrocell (ITM)* blocks. These blocks can generate trace packets during sleep when so enabled. If the TRCENA bit of the Debug Exception and Monitor Control Register is enabled then the power consumption of those blocks is minimized. See *Debug Exception and Monitor Control Register* on page 10-8.

FCLK frequency can be reduced during **SLEEPING** assertion.

———— Note ————

Suppressing **HCLK** using the clock-gating scheme in Figure 7-1 prevents debug accesses. The CoreSight *Debug Ports (DPs)* provide a power up signal that enables the system to bypass the clock-gating logic in Figure 7-1.

7.2.2 SLEEPDEEP

Figure 7-2 shows an example of how to reduce power consumption by stopping the clock controller with **SLEEPDEEP** in the low-power state. When exiting low-power state, the **LOCK** signal indicates that the PLL is stable, and it is safe to enable the Cortex-M3 clock, ensuring that the processor is not re-started until the clocks are stable.

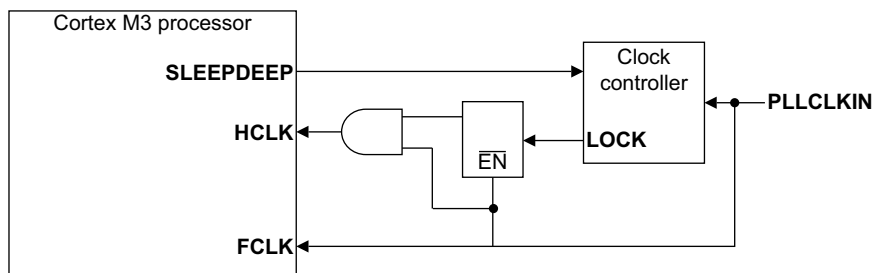


Figure 7-2 SLEEPDEEP power control example

To detect interrupts, the processor must receive the free-running **FCLK** in the low-power state, unless the WIC is enabled. **FCLK** frequency can be reduced during **SLEEPDEEP** assertion.

7.2.3 Extending sleep

You can use the **SLEEPHOLDREQn** and **SLEEPHOLDACKn** signals to extend the sleep state. When the core is asleep with the **SLEEPING** signal raised, **SLEEPHOLDREQn** can be asserted. In the following cycle, **SLEEPHOLDACKn** is asserted to confirm the extension request. When a wake-up event occurs, **SLEEPING** is de-asserted as normal but **SLEEPHOLDACKn** remains asserted and the core remains sleeping. **SLEEPHOLDREQn** must be de-asserted to enable the core to wake up. Halting the core also causes **SLEEPHOLDACKn** to be de-asserted and the sleep mode to be exited as is the case with **SLEEPING**, irrespective of whether **SLEEPHOLDREQn** is asserted or not.

If **SLEEPHOLDREQn** is asserted when **SLEEPING** is not high then the core does not respond and only enters sleep mode when a sleep event occurs.

It is possible to assert **SLEEPING** for a very short time, for example if an interrupt is asserted in parallel to the sleep event being executed. In this case, **SLEEPHOLDREQn** might not be asserted in time for an acknowledge to occur and the sleep mode might not be successfully extended. Your implementation must account for this case and ensure that **SLEEPHOLDREQn** can be de-asserted if the sleep mode is exited without an acknowledge.

If the sleep mode is used to clock gate the **HCLK** signal, as Figure 7-1 on page 7-4 shows, then **SLEEPHOLDACKn** must be inverted and OR'd together with **SLEEPING** to produce the clock gate enable term. **HCLK** must be enabled when the debugger is used. Alternatively a **GATEHCLK** signal, which you can find on the CortexM3Integration level, can be used for gating control of **HCLK**.

7.2.4 Using the Wake-up Interrupt Controller

This subsection describes how to use the *Wake-up Interrupt Controller* (WIC). It contains the following:

- *WIC overview*
- *WIC functionality*.

WIC overview

The Cortex-M3 NVIC has logic dedicated to determining whether at any point in time a newly received interrupt is of higher priority than the current priority and must therefore be taken over the current execution context or priority. This priority scheme also operates during WFE, WFI, and sleep-on-exit to determine when the core must resume execution of instructions after sleeping.

For ultra-low power applications, it is desirable to be able to significantly reduce the dynamic and static power of the processor while in very-deep-sleep modes. This can be achieved by stopping clocks or removing power from the processor, or both. When powered off, the NVIC is unable to prioritize or detect interrupts. This means that knowing when to come out of very-deep-sleep becomes problematic. The *Wake-up Interrupt Controller* (WIC) provides significantly reduced gate count interrupt detection logic that can take over and emulate the full NVIC behavior when correctly primed by the full NVIC on entry to very-deep-sleep. The small size of the WIC ensures that its power requirements fit the budget available while in very-deep-sleep mode enabling it to always remain powered.

Unlike the NVIC, the WIC has no prioritization logic. It implements a rudimentary interrupt masking system, signalling for wake-up as soon as a non-masked interrupt is detected. The WIC contains no programmer's model visible state and is therefore invisible to end users of the device other than through the benefits of reduced power consumption while sleeping.

WIC functionality

You can use the WIC to gate **FCLK** and also to switch off power to the processor, if required. The *Power Management Unit* (PMU) first communicates to the WIC by asserting an enable called **WICENREQ** to the WIC. The WIC then sends a request to the processor to agree to WIC mode sleep. If the processor acknowledges then the WIC

in turn acknowledges the PMU. When all acknowledges are set, then the next SLEEPDEEP mode is agreed to be a WIC mode sleep. Figure 7-3 shows an example of this hand-shaking sequence.

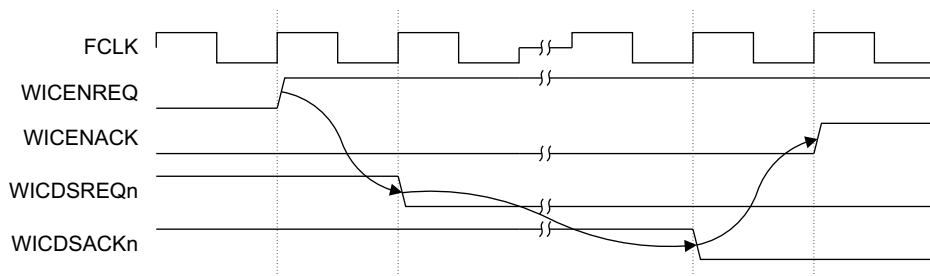


Figure 7-3 WIC mode enable sequence

The next time the deep sleep mode is entered either by WFI, WFE, or sleep-on-exit then the processor loads the WIC with a suitable mask using WICLOAD and WICMASK to enable the required interrupts and events, or both, to cause a wake-up. A logic-1 in the WICSENSE and WICMASK vector, or both, indicates that the WIC must wake up in response to the corresponding WICINT signal. The mask vector is held by the WIC when programmed. WICPEND is a vector of pending interrupts captured by the WIC block. This provides a latched indication of the occurrence of any enabled and detected interrupts which occurred while the NVIC was sleeping. This enables pulse-interrupts to be used in combination with WIC based sleep methods.

The PMU must assert **SLEEPHOLDREQn** to prevent the processor from waking up during a power-down sequence. When it has been acknowledged by **SLEEPHOLDACKn** then the PMU can proceed to power down the system. When the WIC detects a wake-up trigger from an interrupt or an event then it signals to the PMU to power up the processor using the **WAKEUP** pin. When the power is restored, the processor processes the event or interrupts or both given by the **WICPEND** signal that has stored the interrupts that have occurred including ones that do not cause a wake-up event because their priorities are not sufficient. When awake, the processor asserts **WICLEAR** to clear the mask contents stored in the WIC. The processor then proceeds with executing instructions until another WIC mode sleep event occurs.

Figure 7-4 on page 7-8 shows an example of the previously described functionality. It also shows the driving of **ISOLATEN**, **RETAINn** and **PWRDOWN** for use with state retention cells.

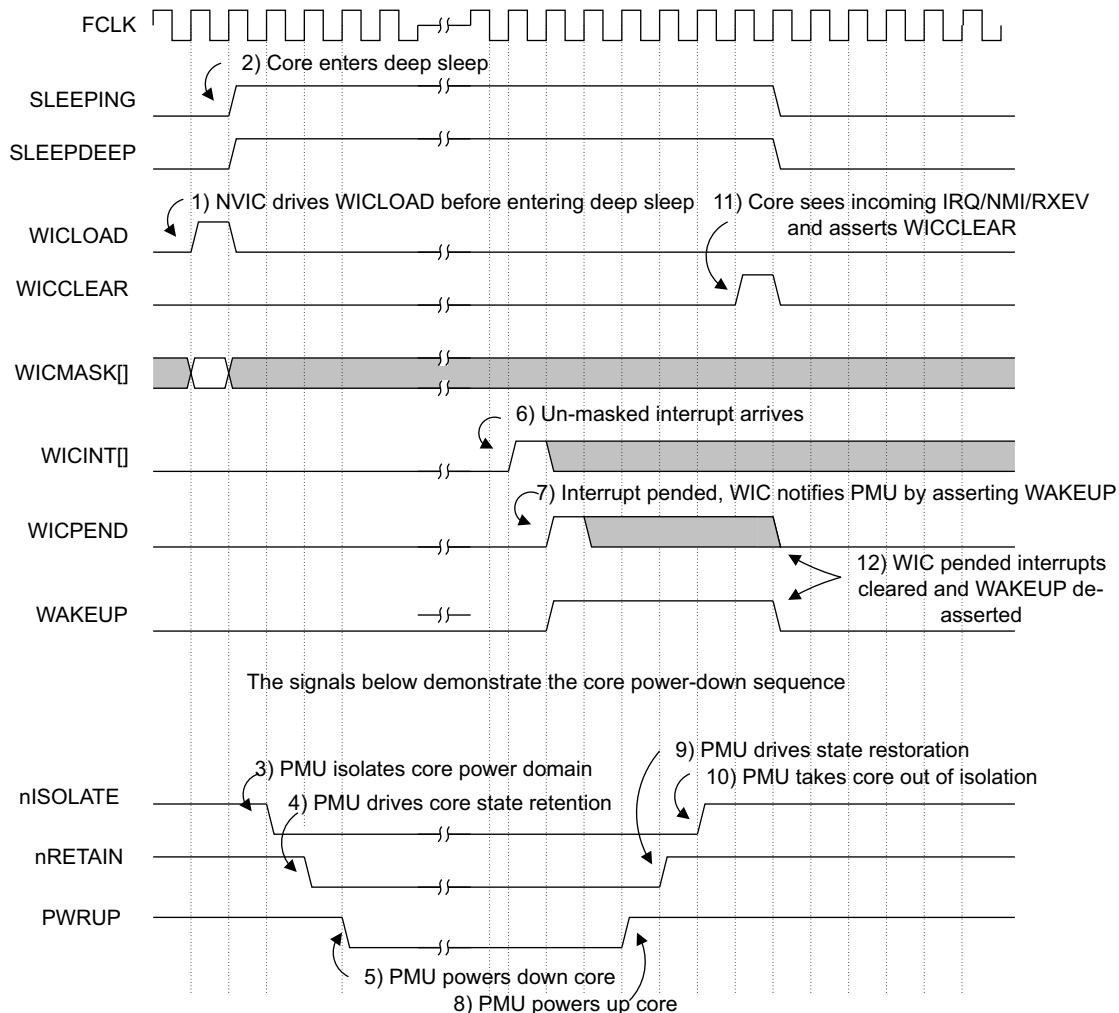


Figure 7-4 Power down timing sequence

Figure 7-5 on page 7-9 shows an example PMU, WIC, and NVIC interconnection with example clamp values for a system using state retention cells. The clamp values have been set to the same value to ease integration. The location of the WICPEND and interrupt OR gates is not important. The clamps are typically inserted by the tools during synthesis.

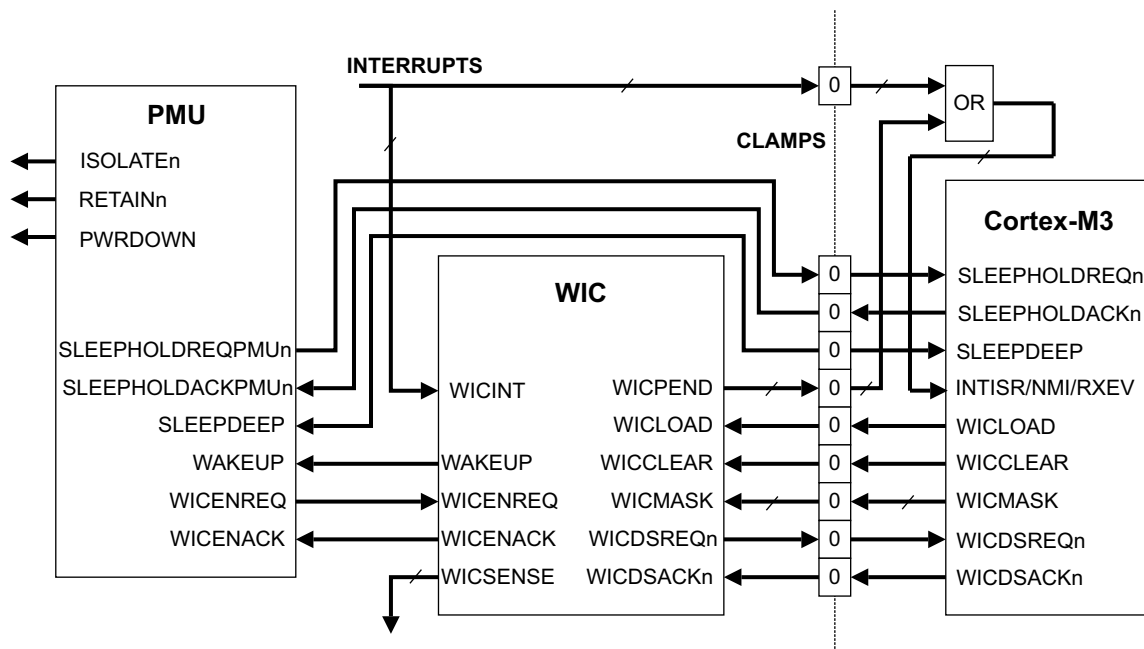


Figure 7-5 PMU, WIC, and Cortex-M3 interconnect

The non-clocked circuitry can use the signals out of the WIC to deduce whether a particular interrupt causes the WIC to generate a WAKEUP request, and to provide alternative power reduction methods not supported by the WIC directly. All WIC interrupt related pins are agnostic as to how many, or what combinations of INTISR, NMI, or RXEV are attached as long as the same offset is used throughout the WIC.

The WIC can be disabled by using **WICDISABLE**, which is a signal indicating that WIC-based SLEEPDEEP must not be entered, or if it has already been entered, that WAKEUP be driven high and the SLEEPDEEP policy revert to non-WIC-based. A debugger must hold this signal high when attached to the system to prevent power isolation during debug.

The sources and causes of wake-up events are implementation defined and the implementation can support any number of signals from two and greater. This enables maximum flexibility over which set of NMI, debug request, interrupts, and RXEV are used as potential wake-up sources.

Note

If the debug logic is included in a powered-down domain, then **nTDOEN** needs to be handled carefully. It cannot be clamped to 0 because this enables it during power down. Either:

- Insert inverters either side of the clamp.
 - Ensure that the external system masks **nTDOEN** when the core is powered down.
 - Clamp **nTDOEN** to 1 during power down.
-

Chapter 8

Nested Vectored Interrupt Controller

This chapter describes the *Nested Vectored Interrupt Controller* (NVIC). It contains the following sections:

- *About the NVIC* on page 8-2
- *NVIC programmer's model* on page 8-3
- *Level versus pulse interrupts* on page 8-43.

8.1 About the NVIC

The NVIC:

- facilitates low-latency exception and interrupt handling
- controls power management
- implements System Control Registers.

The NVIC supports up to 240 dynamically reprioritizable interrupts each with up to 256 levels of priority. The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts. The NVIC maintains knowledge of the stacked (nested) interrupts to enable tail-chaining of interrupts.

You can only fully access the NVIC from privileged mode, but you can pend interrupts in user-mode if you enable the Configuration Control Register (see *Configuration Control Register* on page 8-26). Any other user-mode access causes a bus fault.

All NVIC registers are accessible using byte, halfword, and word unless otherwise stated.

All NVIC registers and system debug registers are little endian regardless of the endianness state of the processor.

Processor exception handling is described in Chapter 5 *Exceptions*.

8.2 NVIC programmer's model

This section lists and describes the NVIC registers. It contains the following:

- *NVIC register map*
- *NVIC register descriptions* on page 8-7.

8.2.1 NVIC register map

Table 8-1 lists the NVIC registers. The System Control space includes the NVIC. The NVIC space is split as follows:

- 0xE000E000 - 0xE000E00F. Interrupt Type Register
- 0xE000E010 - 0xE000E0FF. System Timer
- 0xE000E100 - 0xE000ECFF. NVIC
- 0xE000ED00 - 0xE000ED8F. System Control Block, including:
 - CPUID
 - System control, configuration, and status
 - Fault reporting
- 0xE000EF00 - 0xE000EF0F. Software Trigger Exception Register
- 0xE000EFD0 - 0xE000EFFF. ID space.

Table 8-1 NVIC registers

Name of register	Type	Address	Reset value	Page
Interrupt Control Type Register	Read-only	0xE000E004	a	page 8-7
Auxiliary Control Register	Read/write	0xE000E008	0x00000000	page 8-8
SysTick Control and Status Register	Read/write	0xE000E010	0x00000000	page 8-9
SysTick Reload Value Register	Read/write	0xE000E014	Unpredictable	page 8-10
SysTick Current Value Register	Read/write clear	0xE000E018	Unpredictable	page 8-11
SysTick Calibration Value Register	Read-only	0xE000E01C	STCALIB	page 8-12
Irq 0 to 31 Set Enable Register	Read/write	0xE000E100	0x00000000	page 8-13
.
.
.
Irq 224 to 239 Set Enable Register	Read/write	0xE000E11C	0x00000000	page 8-13

Table 8-1 NVIC registers (continued)

Name of register	Type	Address	Reset value	Page
Irq 0 to 31 Clear Enable Register	Read/write	0xE000E180	0x00000000	page 8-14
.
.
.
Irq 224 to 239 Clear Enable Register	Read/write	0xE000E19C	0x00000000	page 8-14
Irq 0 to 31 Set Pending Register	Read/write	0xE000E200	0x00000000	page 8-15
.
.
.
Irq 224 to 239 Set Pending Register	Read/write	0xE000E21C	0x00000000	page 8-15
Irq 0 to 31 Clear Pending Register	Read/write	0xE000E280	0x00000000	page 8-15
.
.
.
Irq 224 to 239 Clear Pending Register	Read/write	0xE000E29C	0x00000000	page 8-15
Irq 0 to 31 Active Bit Register	Read-only	0xE000E300	0x00000000	page 8-16
.
.
.
Irq 224 to 239 Active Bit Register	Read-only	0xE000E31C	0x00000000	page 8-16
Irq 0 to 3 Priority Register	Read/write	0xE000E400	0x00000000	page 8-17
.
.
.

Table 8-1 NVIC registers (continued)

Name of register	Type	Address	Reset value	Page
Irq 224 to 239 Priority Register	Read/write	0xE000E4EC	0x00000000	page 8-17
CPUID Base Register	Read-only	0xE000ED00	0x412FC230	page 8-18
Interrupt Control State Register	Read/write or read-only	0xE000ED04	0x00000000	page 8-19
Vector Table Offset Register	Read/write	0xE000ED08	0x00000000	page 8-21
Application Interrupt/Reset Control Register	Read/write	0xE000ED0C	0x00000000 ^b	page 8-22
System Control Register	Read/write	0xE000ED10	0x00000000	page 8-25
Configuration Control Register	Read/write	0xE000ED14	0x00000000	page 8-26
System Handlers 4-7 Priority Register	Read/write	0xE000ED18	0x00000000	page 8-28
System Handlers 8-11 Priority Register	Read/write	0xE000ED1C	0x00000000	page 8-28
System Handlers 12-15 Priority Register	Read/write	0xE000ED20	0x00000000	page 8-28
System Handler Control and State Register	Read/write	0xE000ED24	0x00000000	page 8-29
Configurable Fault Status Registers	Read/write	0xE000ED28	0x00000000	page 8-32
Hard Fault Status Register	Read/write	0xE000ED2C	0x00000000	page 8-37
Debug Fault Status Register	Read/write	0xE000ED30	0x00000000	page 8-38
Mem Manage Address Register	Read/write	0xE000ED34	Unpredictable	page 8-40
Bus Fault Address Register	Read/write	0xE000ED38	Unpredictable	page 8-41
Auxiliary Fault Status Register	Read/write	0xE000ED3C	0x00000000	page 8-41
PFR0: Processor Feature register0	Read-only	0xE000ED40	0x00000030	-
PFR1: Processor Feature register1	Read-only	0xE000ED44	0x00000200	-
DFR0: Debug Feature register0	Read-only	0xE000ED48	0x00100000	-
AFR0: Auxiliary Feature register0	Read-only	0xE000ED4C	0x00000000	-
MMFR0: Memory Model Feature register0	Read-only	0xE000ED50	0x00000030	-
MMFR1: Memory Model Feature register1	Read-only	0xE000ED54	0x00000000	-
MMFR2: Memory Model Feature register2	Read-only	0xE000ED58	0x00000000	-

Table 8-1 NVIC registers (continued)

Name of register	Type	Address	Reset value	Page
MMFR3: Memory Model Feature register3	Read-only	0xE000ED5C	0x00000000	-
ISAR0: ISA Feature register0	Read-only	0xE000ED60	0x01141110	-
ISAR1: ISA Feature register1	Read-only	0xE000ED64	0x02111000	-
ISAR2: ISA Feature register2	Read-only	0xE000ED68	0x21112231	-
ISAR3: ISA Feature register3	Read-only	0xE000ED6C	0x01111110	-
ISAR4: ISA Feature register4	Read-only	0xE000ED70	0x01310102	-
Software Trigger Interrupt Register	Write Only	0xE000EF00	-	page 8-42
Peripheral identification register (PID4)	Read-only	0xE000EFD0	0x04	-
Peripheral identification register (PID5)	Read-only	0xE000EFD4	0x00	-
Peripheral identification register (PID6)	Read-only	0xE000EFD8	0x00	-
Peripheral identification register (PID7)	Read-only	0xE000EFD C	0x00	-
Peripheral identification register Bits 7:0 (PID0)	Read-only	0xE000EFE0	0x00	-
Peripheral identification register Bits 15:8 (PID1)	Read-only	0xE000EFE4	0xB0	-
Peripheral identification register Bits 23:16 (PID2)	Read-only	0xE000EFE8	0x2B	-
Peripheral identification register Bits 31:24 (PID3)	Read-only	0xE000EFEC	0x00	-
Component identification register Bits 7:0 (CID0)	Read Only	0xE000EFF0	0x0D	-
Component identification register Bits 15:8 (CID1)	Read-only	0xE000EFF4	0xE0	-
Component identification register Bits 23:16 (CID2)	Read-only	0xE000EFF8	0x05	-
Component identification register Bits 31:24 (CID3)	Read-only	0xE000EFFC	0xB1	-

a. Reset value depends on the number of interrupts defined.

b. Bits [10:8] are reset. The ENDIANESS bit, bit [15], is set at reset by the sampling of **BIGEND**.

8.2.2 NVIC register descriptions

The sections that follow describe how to use the NVIC registers.

———— **Note** —————

The *Memory Protection Unit* (MPU) registers, and the debug registers are described in Chapter 9 *Memory Protection Unit* and Chapter 10 *Core Debug* respectively.

Interrupt Controller Type Register

Read the Interrupt Controller Type Register to see the number of interrupt lines that the NVIC supports.

The register address, access type, and Reset state are:

Address	0xE000E004
Access	Read-only
Reset state	Depends on the number of interrupts defined in this processor implementation.

Figure 8-1 shows the bit assignments of the Interrupt Controller Type Register.

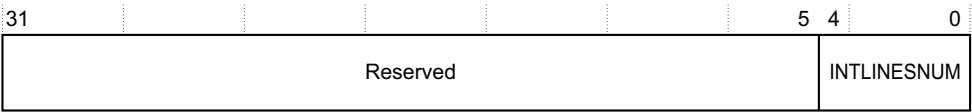


Figure 8-1 Interrupt Controller Type Register bit assignments

Table 8-2 describes the bit assignments of the Interrupt Controller Type Register.

Table 8-2 Interrupt Controller Type Register bit assignments

Bits	Field	Function
[31:5]	-	Reserved.
[4:0]	INTLINESNUM	Total number of interrupt lines in groups of 32: b00000 = 0...32 ^a b00001 = 33...64 b00010 = 65...96 b00011 = 97...128 b00100 = 129...160 b00101 = 161...192 b00110 = 193...224 b00111 = 225...256 ^a

a. The processor only supports between 1 and 240 external interrupts.

Auxiliary Control Register

Use the Auxiliary Control Register to disable certain aspects of functionality within the processor.

The register address, access type, and Reset state are:

Address 0xE000E008
Access Read/write
Reset state 0x00000000

Figure 8-2 shows the bit assignments of the Auxiliary Control Register.

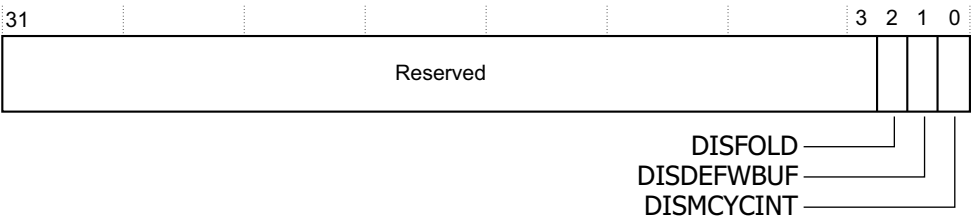


Figure 8-2 Auxiliary Control Register bit assignments

Table 8-3 describes the bit assignments of the Auxiliary Control Register.

Table 8-3 Auxiliary Control Register bit assignments

Bits	Field	Function
[31:3]	-	Reserved
[2]	DISFOLD	Disables IT folding.
[1]	DISDEFWBUF	Disables write buffer use during default memory map accesses. This causes all bus faults to be precise bus faults but decreases the performance of the processor because the stores to memory have to complete before the next instruction can be executed.
[0]	DISMCYCINT	Disables interruption of multi-cycle instructions. This increases the interrupt latency of the processor because LDM/STM completes before interrupt stacking occurs.

SysTick Control and Status Register

Use the SysTick Control and Status Register to enable the SysTick features.

The register address, access type, and Reset state are:

Address 0xE000E010
Access Read/write
Reset state 0x00000000

Figure 8-3 shows the bit assignments of the SysTick Control and Status Register.

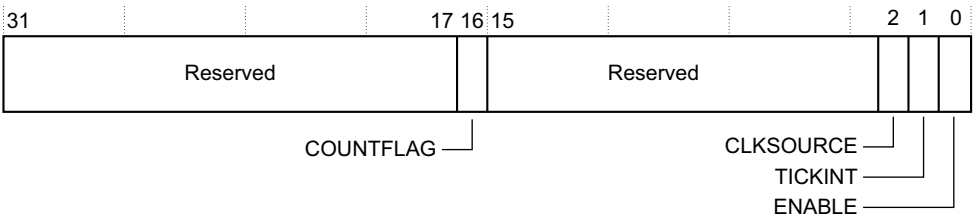


Figure 8-3 SysTick Control and Status Register bit assignments

Table 8-4 describes the bit assignments of the SysTick Control and Status register.

Table 8-4 SysTick Control and Status Register bit assignments

Bits	Field	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read. Clears on read by application of any part of the SysTick Control and Status Register. If read by the debugger using the DAP, this bit is cleared on read-only if the MasterType bit in the AHB-AP Control Register is set to 0. Otherwise, the COUNTFLAG bit is not changed by the debugger read.
[2]	CLKSOURCE	0 = external reference clock. 1 = core clock. If no reference clock is provided, it is held at 1 and so gives the same time as the core clock. The core clock must be at least 2.5 times faster than the reference clock. If it is not, the count values are Unpredictable.
[1]	TICKINT	1 = counting down to 0 pends the SysTick handler. 0 = counting down to 0 does not pend the SysTick handler. Software can use the COUNTFLAG to determine if ever counted to 0.
[0]	ENABLE	1 = counter operates in a multi-shot way. That is, counter loads with the Reload value and then begins counting down. On reaching 0, it sets the COUNTFLAG to 1 and optionally pends the SysTick handler, based on TICKINT. It then loads the Reload value again, and begins counting. 0 = counter disabled.

SysTick Reload Value Register

Use the SysTick Reload Value Register to specify the start value to load into the current value register when the counter reaches 0. It can be any value between 1 and 0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick interrupt and COUNTFLAG are activated when counting from 1 to 0.

Therefore, as a multi-shot timer, repeated over and over, it fires every N+1 clock pulse, where N is any value from 1 to 0x00FFFFFF. So, if the tick interrupt is required every 100 clock pulses, 99 must be written into the RELOAD. If a new value is written on each tick interrupt, so treated as single shot, then the actual count down must be written. For example, if a tick is next required after 400 clock pulses, 400 must be written into the RELOAD.

The register address, access type, and Reset state are:

Address 0xE000E014
Access Read/write

Copyright © 2005-2008 ARM Limited. All rights reserved.
Non-Confidential

8-11

Figure 8-4 SysTick Reload Value Register bit assignments

Table 8-5 SysTick Reload Value Register bit assignments

Figure 8-5 SysTick Current Value Register bit assignments

Table 8-6 describes the bit assignments of the SysTick Current Value Register.

Table 8-6 SysTick Current Value Register bit assignments

Bits	Field	Function
[31:24]	-	Reserved
[23:0]	CURRENT	Current value at the time the register is accessed. No read-modify-write protection is provided, so change with care. This register is write-clear. Writing to it with any value clears the register to 0. Clearing this register also clears the COUNTFLAG bit of the SysTick Control and Status Register.

SysTick Calibration Value Register

Use the SysTick Calibration Value Register to enable software to scale to any required speed using divide and multiply.

The register address, access type, and Reset state are:

Address 0xE000E01C
Access Read
Reset state STCALIB

Figure 8-6 describes the bit assignments of the SysTick Calibration Value Register.

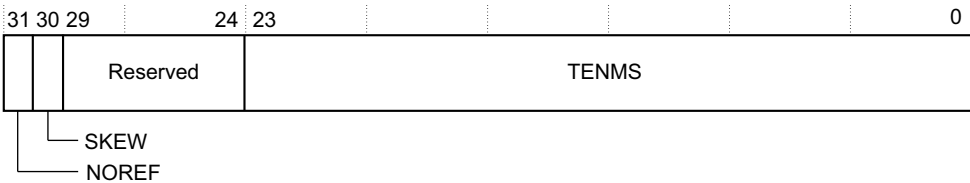


Figure 8-6 SysTick Calibration Value Register bit assignments

Table 8-7 describes the bit assignments of the SysTick Calibration Value Register.

Table 8-7 SysTick Calibration Value Register bit assignments

Bits	Field	Function
[31]	NOREF	1 = the reference clock is not provided.

Table 8-7 SysTick Calibration Value Register bit assignments (continued)

Bits	Field	Function
[30]	SKEW	1 = the calibration value is not exactly 10ms because of clock frequency. This could affect its suitability as a software real time clock.
[29:24]	-	Reserved
[23:0]	TENMS	This value is the Reload value to use for 10ms timing. Depending on the value of SKEW, this might be exactly 10ms or might be the closest value. If this reads as 0, then the calibration value is not known. This is probably because the reference clock is an unknown input from the system or scalable dynamically.

Interrupt Set-Enable Registers

Use the Interrupt Set-Enable Registers to:

- enable interrupts
- determine which interrupts are currently enabled.

Each bit in the register corresponds to one of 32 interrupts. Setting a bit in the Interrupt Set-Enable Register enables the corresponding interrupt.

When the enable bit of a pending interrupt is set, the processor activates the interrupt based on its priority. When the enable bit is clear, asserting its interrupt signal pends the interrupt, but it is not possible to activate the interrupt, regardless of its priority. Therefore, a disabled interrupt can serve as a latched general-purpose I/O bit. You can read it and clear it without invoking an interrupt.

Clear an Interrupt Set-Enable Register bit by writing a 1 to the corresponding bit in the Interrupt Clear-Enable Register (see *Interrupt Clear-Enable Registers* on page 8-14).

————— Note —————

Clearing an Interrupt Set-Enable Register bit does not affect currently active interrupts. It only prevents new activations.

The register address, access type, and Reset state are:

Address	0xE000E100–0xE000E11C
Access	Read/write
Reset state	0x00000000

Table 8-8 describes the field of the Interrupt Set-Enable Register.

Table 8-8 Interrupt Set-Enable Register bit assignments

Bits	Field	Function
[31:0]	SETENA	Interrupt set enable bits. For write operation: 1 = enable interrupt 0 = no effect. For read operation: 1 = enable interrupt 0 = disable interrupt Writing 0 to a SETENA bit has no effect. Reading the bit returns its current enable state. Reset clears the SETENA fields.

Interrupt Clear-Enable Registers

Use the Interrupt Clear-Enable Registers to:

- disable interrupts
- determine which interrupts are currently disabled.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Clear-Enable Register bit disables the corresponding interrupt.

The register address, access type, and Reset state are:

Address	0xE000E180-0xE000E19C
Access	Read/write
Reset state	0x00000000

Table 8-9 describes the field of the Interrupt Clear-Enable Register.

Table 8-9 Interrupt Clear-Enable Register bit assignments

Bits	Field	Function
[31:0]	CLRENA	Interrupt clear-enable bits. For write operation: 1 = disable interrupt 0 = no effect. For read operation: 1 = enable interrupt 0 = disable interrupt. Writing 0 to a CLRENA bit has no effect. Reading the bit returns its current enable state.

Interrupt Set-Pending Register

Use the Interrupt Set-Pending Register to:

- force interrupts into the pending state
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Set-Pending Register bit pends the corresponding interrupt.

Clear an Interrupt Set-Pending Register bit by writing a 1 to the corresponding bit in the Interrupt Clear-Pending Register (see *Interrupt Clear-Pending Register*). Clearing the Interrupt Set-Pending Register bit puts the interrupt into the non-pended state.

Note

Writing to the Interrupt Set-Pending Register has no affect on an interrupt that is already pending or is disabled.

The register address, access type, and Reset state are:

Address	0xE000E200-0xE000E21C
Access	Read/write
Reset state	0x00000000

Table 8-10 describes the field of the Interrupt Set-Pending Register.

Table 8-10 Interrupt Set-Pending Register bit assignments

Bits	Field	Function
[31:0]	SETPEND	Interrupt set-pending bits: 1 = pend the corresponding interrupt 0 = corresponding interrupt not pending. Writing 0 to a SETPEND bit has no effect. Reading the bit returns its current state.

Interrupt Clear-Pending Register

Use the Interrupt Clear-Pending Register to:

- clear pending interrupts
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Clear-Pending Register bit puts the corresponding pending interrupt in the inactive state.

Note

Writing to the Interrupt Clear-Pending Register has no effect on an interrupt that is active unless it is also pending.

The register address, access type, and Reset state are:

Address	0xE000E280-0xE000E29C
Access	Read/write
Reset state	0x00000000

Table 8-11 describes the field of the Interrupt Clear-Pending Registers.

Table 8-11 Interrupt Clear-Pending Registers bit assignments

Bits	Field	Function
[31:0]	CLRPEND	Interrupt clear-pending bits: 1 = clear pending interrupt 0 = do not clear pending interrupt. Writing 0 to a CLRPEND bit has no effect. Reading the bit returns its current state.

Active Bit Register

Read the Active Bit Register to determine which interrupts are active. Each flag in the register corresponds to one of the 32 interrupts.

The register address, access type, and Reset state are:

Address	0xE000E300-0xE000E31C
Access	Read-only
Reset state	0x00000000

Table 8-12 describes the field of the Active Bit Register.

Table 8-12 Active Bit Register bit assignments

Bits	Field	Function
[31:0]	ACTIVE	Interrupt active flags: 1 = interrupt active or pre-empted and stacked 0 = interrupt not active or stacked.

Interrupt Priority Registers

Use the Interrupt Priority Registers to assign a priority from 0 to 255 to each of the available interrupts. 0 is the highest priority, and 255 is the lowest.

The priority registers are stored with the implemented values first. This means that if there are four bits of priority, the priority value is stored in bits [7:4] of the byte. However, if there are three bits of priority, the priority value is stored in bits [7:5] of the byte. This means that an application can work even if it does not know how many priorities are possible.

The register address, access type, and Reset state are:

Address 0xE000E400-0xE000E41F

Access Read/write

Reset state 0x00000000

Figure 8-7 shows the bit assignments of Interrupt Priority Registers 0-7 for interrupts 0-31.

	31	24	23	16	15	8	7	0
E000E400	PRI_3				PRI_2			
E000E404	PRI_7				PRI_6			
E000E408	PRI_11				PRI_10			
E000E40C	PRI_15				PRI_14			
E000E410	PRI_19				PRI_18			
E000E414	PRI_23				PRI_22			
E000E418	PRI_27				PRI_26			
E000E41C	PRI_31				PRI_30			

Figure 8-7 Interrupt Priority Registers 0-31 bit assignments

The lower PRI_*n* bits can specify subpriorities for priority grouping. See *Exception priority* on page 5-6.

Table 8-13 describes the bit assignments of the Interrupt Priority Registers, where *n* specifies the number of interrupts and is greater than 0 and less than or equal to 240.

Table 8-13 Interrupt Priority Registers 0-31 bit assignments

Bits	Field	Function
[7:0]	PRI_ <i>n</i>	Priority of interrupt <i>n</i>

CPU ID Base Register

Read the CPU ID Base Register to determine:

- the ID number of the processor core
- the version number of the processor core
- the implementation details of the processor core.

The register address, access type, and Reset state are:

Address 0xE000ED00
Access Read-only
Reset state 0x412FC230

Figure 8-8 shows the bit assignments of the CPUID Base Register.

31	24	23	20	19	16	15	4	3	0
IMPLEMENTER				VARIANT		Constant	PARTNO		REVISION

Figure 8-8 CPUID Base Register bit assignments

Table 8-14 describes the bit assignments of the CPUID Base Register.

Table 8-14 CPUID Base Register bit assignments

Bits	Field	Function
[31:24]	IMPLEMENTER	Implementer code. ARM is 0x41
[23:20]	VARIANT	Implementation defined variant number.

Table 8-14 CPUID Base Register bit assignments (continued)

Bits	Field	Function
[19:16]	Constant	Reads as 0xF
[15:4]	PARTNO	Number of processor within family: [11:10] b11 = Cortex family [9:8] b00 = version [7:6] b00 = reserved [5:4] b10 = M (v7-M) [3:0] X = family member. Cortex-M3 family is b0011.
[3:0]	REVISION	Implementation defined revision number.

Interrupt Control State Register

Use the Interrupt Control State Register to:

- set a pending *Non-Maskable Interrupt* (NMI)
- set or clear a pending SVC
- set or clear a pending SysTick
- check for pending exceptions
- check the vector number of the highest priority pended exception
- check the vector number of the active exception.

The register address, access type, and Reset state are:

Address 0xE000ED04

Access Read/write or read-only

Reset state 0x00000000

Figure 8-9 on page 8-20 shows the bit assignments of the Interrupt Control State Register.

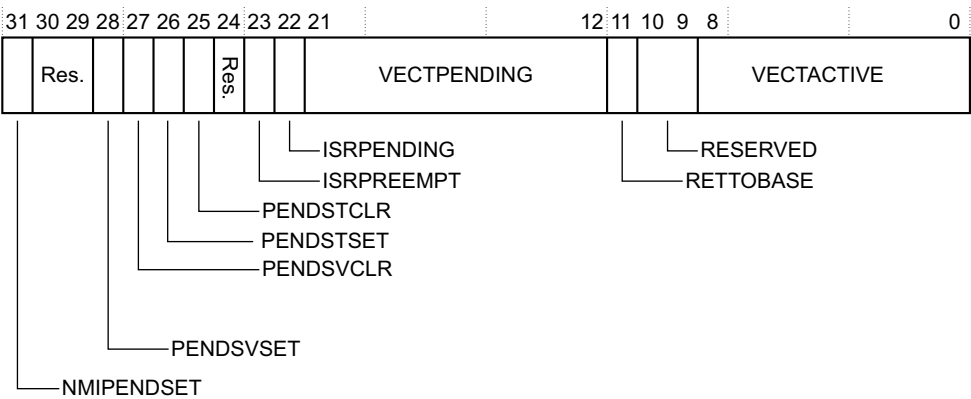


Figure 8-9 Interrupt Control State Register bit assignments

Table 8-15 describes the bit assignments of the Interrupt Control State Register.

Table 8-15 Interrupt Control State Register bit assignments

Bits	Field	Type	Function
[31]	NMIPENDSET	Read/write	Set pending NMI bit: 1 = set pending NMI 0 = do not set pending NMI. NMIPENDSET pends and activates an NMI. Because NMI is the highest-priority interrupt, it takes effect as soon as it registers.
[30:29]	-	-	Reserved.
[28]	PENDSVSET	Read/write	Set pending pendSV bit: 1 = set pending pendSV 0 = do not set pending pendSV.
[27]	PENDSVCLR	Write-only	Clear pending pendSV bit: 1 = clear pending pendSV 0 = do not clear pending pendSV.
[26]	PENDSTSET	Read/write	Set a pending SysTick bit 1 = set pending SysTick 0 = do not set pending SysTick.
[25]	PENDSTCLR	Write-only	Clear pending SysTick bit: 1 = clear pending SysTick 0 = do not clear pending SysTick.

Table 8-15 Interrupt Control State Register bit assignments (continued)

Bits	Field	Type	Function
[24]	-	-	Reserved
[23]	ISRPREEMPT	Read-only	You must only use this at debug time. It indicates that a pending interrupt becomes active in the next running cycle. If C_MASKINTS is clear in the Debug Halting Control and Status Register, the interrupt is serviced.
[22]	ISRPENDING	Read-only	Interrupt pending flag. Excludes NMI and Faults: 1 = interrupt pending 0 = interrupt not pending.
[21:12]	VECTPENDING	Read-only	Pending ISR number field. VECTPENDING contains the interrupt number of the highest priority pending ISR.
[11]	RETTOBASE	Read-only	This bit is 1 when the set of all active exceptions minus the IPSR_current_exception yields the empty set.
[10]	-	-	Reserved.
[9]	-	-	Reserved
[8:0]	VECTACTIVE	Read-only	Active ISR number field. VECTACTIVE contains the interrupt number of the currently running ISR, including NMI and Hard Fault. A shared handler can use VECTACTIVE to determine which interrupt invoked it. You can subtract 16 from the VECTACTIVE field to index into the Interrupt Clear/Set Enable, Interrupt Clear Pending/SetPending and Interrupt Priority Registers. INTISR[0] has vector number 16. Reset clears the VECTACTIVE field.

Vector Table Offset Register

Use the Vector Table Offset Register to determine:

- if the vector table is in RAM or code memory
- the vector table offset.

The register address, access type, and Reset state are:

Address 0xE000ED08

Access Read/write

Reset state 0x00000000

Figure 8-10 on page 8-22 shows the bit assignments of the Vector Table Offset Register.

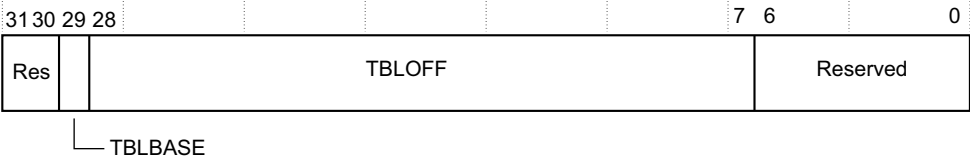


Figure 8-10 Vector Table Offset Register bit assignments

Table 8-16 describes the bit assignments of the Vector Table Offset Register.

Table 8-16 Vector Table Offset Register bit assignments

Bits	Field	Function
[31:30]	-	Reserved, RAZ/WI
[29]	TBLBASE	Table base is in Code (0) or RAM (1)
[28:7]	TBLOFF	Vector table base offset field. Contains the offset of the table base from the bottom of the SRAM or CODE space.
[6:0]	-	Reserved, RAZ/WI

The Vector Table Offset Register positions the vector table in CODE or SRAM space. The default, on reset, is 0 (CODE space). When setting a position, the offset must be aligned based on the number of exceptions in the table. This means that the minimum alignment is 32 words that you can use for up to 16 interrupts. For more interrupts, you must adjust the alignment by rounding up to the next power of two. For example, if you require 21 interrupts, the alignment must be on a 64-word boundary because table size is 37 words, next power of two is 64.

Note

Table alignment requirements mean that bits [6:0] of the table offset are always zero. TBLBASE and TBLOFF are combined with 7'b0000000 to construct the complete vector table base offset value.

Application Interrupt and Reset Control Register

Use the Application Interrupt and Reset Control Register to:

- determine data endianness
- clear all active state information for debug or to recover from a hard failure
- execute a system reset
- alter the priority grouping position (binary point).

The register address, access type, and Reset state are:

Address 0xE000ED0C
Access Read/write
Reset state 0x00000000

Figure 8-11 shows the bit assignments of the Application Interrupt and Reset Control Register.

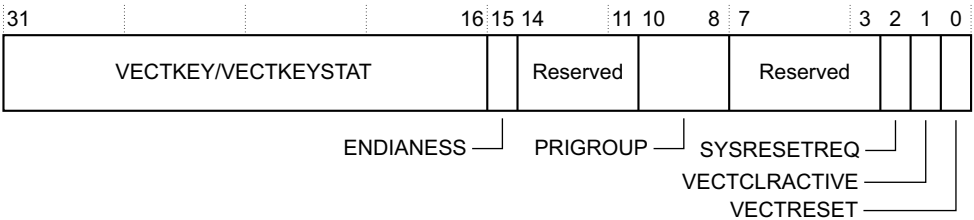


Figure 8-11 Application Interrupt and Reset Control Register bit assignments

Table 8-17 describes the bit assignments of the Application Interrupt and Reset Control Register.

Table 8-17 Application Interrupt and Reset Control Register bit assignments

Bits	Field	Function
[31:16]	VECTKEY	Register key. Writing to this register requires 0x5FA in the VECTKEY field. Otherwise the write value is ignored.
[31:16]	VECTKEYSTAT	Reads as 0xFA05.
[15]	ENDIANESS	Data endianness bit: 1 = big endian 0 = little endian. ENDIANESS is sampled from the BIGEND input port during reset. You cannot change ENDIANESS outside of reset.
[14:11]	-	Reserved
[10:8]	PRIGROUP	Interrupt priority grouping field:

Table 8-17 Application Interrupt and Reset Control Register bit assignments (continued)

Bits	Field	Function
		<p>PRIGROUP 0 Split of pre-emption priority from subpriority</p> <p>1 7.1 indicates seven bits of pre-emption priority, one bit of subpriority</p> <p>2 6.2 indicates six bits of pre-emption priority, two bits of subpriority</p> <p>3 5.3 indicates five bits of pre-emption priority, three bits of subpriority</p> <p>4 4.4 indicates four bits of pre-emption priority, four bits of subpriority</p> <p>5 3.5 indicates three bits of pre-emption priority, five bits of subpriority</p> <p>6 2.6 indicates two bits of pre-emption priority, six bits of subpriority</p> <p>7 1.7 indicates one bit of pre-emption priority, seven bits of subpriority</p> <p>0.8 indicates no pre-emption priority, eight bits of subpriority.</p> <p>PRIGROUP field is a binary point position indicator for creating subpriorities for exceptions that share the same pre-emption level. It divides the PRI_n field in the Interrupt Priority Register into a pre-emption level and a subpriority level. The binary point is a left-of value. This means that the PRIGROUP value represents a point starting at the left of the <i>Least Significant Bit</i> (LSB). This is bit [0] of 7:0.</p> <p>The lowest value might not be 0 depending on the number of bits allocated for priorities, and implementation choices.</p>
[7:3]	-	Reserved.
[2]	SYSRESETREQ	Causes a signal to be asserted to the outer system that indicates a reset is requested. Intended to force a large system reset of all major components except for debug. Setting this bit does not prevent Halting Debug from running.
[1]	VECTCLRACTIVE	<p>Clear active vector bit:</p> <p>1 = clear all state information for active NMI, fault, and interrupts</p> <p>0 = do not clear.</p> <p>It is the responsibility of the application to reinitialize the stack.</p> <p>The VECTCLRACTIVE bit is for returning to a known state during debug. The VECTCLRACTIVE bit self-clears.</p> <p>IPSR is not cleared by this operation. So, if used by an application, it must only be used at the base level of activation, or within a system handler whose active bit can be set.</p>
[0]	VECTRESET	<p>System Reset bit. Resets the system, with the exception of debug components:</p> <p>1 = reset system</p> <p>0 = do not reset system.</p> <p>The VECTRESET bit self-clears. Reset clears the VECTRESET bit.</p> <p>For debugging, only write this bit when the core is halted.</p>

Note

SYSRESETREQ is cleared by a system reset, which means that asserting **VECTRESET** at the same time might cause **SYSRESETREQ** to be cleared in the same cycle as it is written to. This might prevent the external system from seeing **SYSRESETREQ**. It is therefore recommended that **VECTRESET** and **SYSRESETREQ** be used exclusively and never both written to 1 at the same time.

System Control Register

Use the System Control Register for power-management functions:

- signal to the system when the processor can enter a low power state
- control how the processor enters and exits low power states.

The register address, access type, and Reset state are:

Address 0xE000ED10
Access Read/write
Reset state 0x00000000

Figure 8-12 shows the bit assignments of the System Control Register.

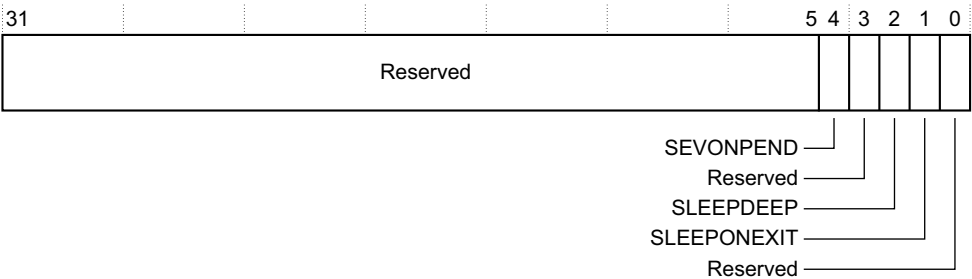


Figure 8-12 System Control Register bit assignments

Table 8-18 describes the bit assignments of the System Control Register.

Table 8-18 System Control Register bit assignments

Bits	Field	Function
[31:5]	-	Reserved.
[4]	SEVONPEND	When enabled, this causes WFE to wake up when an interrupt moves from inactive to pended. Otherwise, WFE only wakes up from an event signal, external and SEV instruction generated. The event input, RXEV, is registered even when not waiting for an event, and so effects the next WFE.
[3]	-	Reserved.
[2]	SLEEPDEEP	Sleep deep bit: 1 = indicates to the system that Cortex-M3 clock can be stopped. Setting this bit causes the SLEEPDEEP port to be asserted when the processor can be stopped. 0 = not OK to turn off system clock. For more information about the use of SLEEPDEEP , see Chapter 7 <i>Power Management</i> .
[1]	SLEEPONEXIT	Sleep on exit when returning from Handler mode to Thread mode: 1 = sleep on ISR exit. 0 = do not sleep when returning to Thread mode. Enables interrupt driven applications to avoid returning to empty main application.
[0]	-	Reserved.

Configuration Control Register

Use the Configuration Control Register to:

- enable NMI, Hard Fault and FAULTMASK to ignore bus fault
- trap divide by zero, and unaligned accesses
- enable user access to the Software Trigger Exception Register
- control entry to Thread Mode.

The register address, access type, and Reset state are:

Address 0xE000ED14
Access Read/write
Reset state 0x00000200

———— **Note** —————

STKALIGN can be reset to 0 or 1 depending on the value of STKALIGNINIT.

Figure 8-13 shows the bit assignments of the Configuration Control Register.

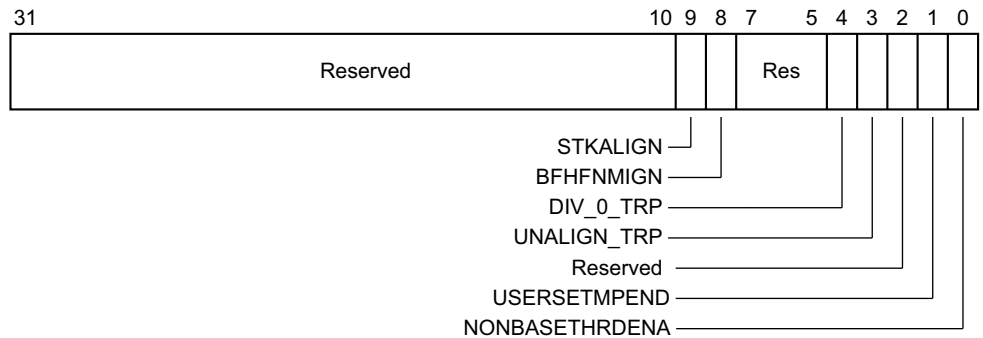


Figure 8-13 Configuration Control Register bit assignments

Table 8-19 describes the bit assignments of the Configuration Control Register.

Table 8-19 Configuration Control Register bit assignments

Bits	Field	Function
[31:10]	–	Reserved.
[9]	STKALIGN	1 = on exception entry, the SP used prior to the exception is adjusted to be 8-byte aligned and the context to restore it is saved. The SP is restored on the associated exception return. 0 = only 4-byte alignment is guaranteed for the SP used prior to the exception on exception entry.
[8]	BFHFNMIGN	When enabled, this causes handlers running at priority -1 and -2 (Hard Fault, NMI, and FAULTMASK escalated handlers) to ignore Data Bus faults caused by load and store instructions. When disabled, these bus faults cause a lock-up. You must only use this enable with extreme caution. All data bus faults are ignored – you must only use it when the handler and its data are in absolutely safe memory. Its normal use is to probe system devices and bridges to detect control path problems and fix them.
[7:5]	–	Reserved.
[4]	DIV_0_TRP	Trap on Divide by 0. This enables faulting/halting when an attempt is made to divide by 0. The relevant Usage Fault Status Register bit is DIVBYZERO, see <i>Usage Fault Status Register</i> on page 8-35.

Table 8-19 Configuration Control Register bit assignments (continued)

Bits	Field	Function
[3]	UNALIGN_TRP	Trap for unaligned access. This enables faulting/halting on any unaligned half or full word access. Unaligned load-store multiples always fault. The relevant Usage Fault Status Register bit is UNALIGNED, see <i>Usage Fault Status Register</i> on page 8-35.
[2]	-	Reserved.
[1]	USERSETMPEND	If written as 1, enables user code to write the Software Trigger Interrupt register to trigger (pend) a Main exception, which is one associated with the Main stack pointer.
[0]	NONEBASETHRDENA	When 0, default, It is only possible to enter Thread mode when returning from the last exception. When set to 1, Thread mode can be entered from any level in Handler mode by controlled return value.

System Handler Priority Registers

Use the three System Handler Priority Registers to prioritize the following system handlers:

- memory manage
- bus fault
- usage fault
- debug monitor
- SVC
- SysTick
- PendSV.

System handlers are a special class of exception handler that can have their priority set to any of the priority levels. Most can be masked on (enabled) or off (disabled). When disabled, the fault is always treated as a Hard Fault.

The register addresses, access types, and Reset states are:

Address 0xE000ED18, 0xE000ED1C , 0xE000ED20
Access Read/write
Reset state 0x00000000

Figure 8-14 on page 8-29 shows the bit assignments of the System Handler Priority Registers.

	31	24	23	16	15	8	7	0
E000ED18	PRI_7		PRI_6		PRI_5		PRI_4	
E000ED1C	PRI_11		PRI_10		PRI_9		PRI_8	
E000ED20	PRI_15		PRI_14		PRI_13		PRI_12	

Figure 8-14 System Handler Priority Registers bit assignments

Table 8-20 describes the bit assignments of the System Handler Priority Registers.

Table 8-20 System Handler Priority Registers bit assignments

Bits	Field	Function
[31:24]	PRI_N3	Priority of system handler 7, 11, and 15. Reserved, SVCall, and SysTick.
[23:16]	PRI_N2	Priority of system handler 6, 10, and 14. Usage Fault, reserved, and PendSV.
[15:8]	PRI_N1	Priority of system handler 5, 9, and 13, Bus Fault, reserved, and reserved.
[7:0]	PRI_N	Priority of system handler 4, 8, and 12. Mem Manage, reserved, and Debug Monitor.

System Handler Control and State Register

Use the System Handler Control and State Register to:

- enable or disable the system handlers
- determine the pending status of bus fault, mem manage fault, and SVC
- determine the active status of the system handlers.

If a fault condition occurs while its fault handler is disabled, the fault escalates to a Hard Fault.

The register address, access type, and Reset state are:

Address 0xE000ED24
Access Read/write
Reset state 0x00000000

Figure 8-15 on page 8-30 shows the bit assignments of the System Handler and State Control Register.

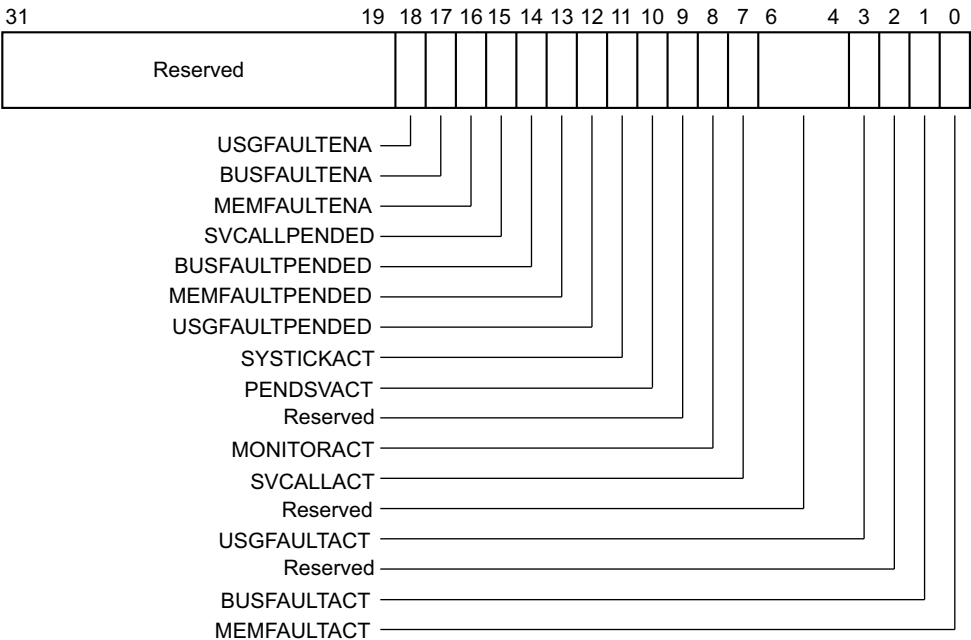


Figure 8-15 System Handler Control and State Register bit assignments

Table 8-21 describes the bit assignments of the System Handler Control Register.

Table 8-21 System Handler Control and State Register bit assignments

Bits	Field	Function
[31:19]	-	Reserved
[18]	USGFAULTENA	Set to 0 to disable, else 1 for enabled.
[17]	BUSFAULTENA	Set to 0 to disable, else 1 for enabled.
[16]	MEMFAULTENA	Set to 0 to disable, else 1 for enabled.
[15]	SVCALLPENDEd	Reads as 1 if SVCall is pended.
[14]	BUSFAULTPENDEd	Reads as 1 if BusFault is pended.
[13]	MEMFAULTPENDEd	Reads as 1 if MemManage is pended.
[12]	USGFAULTPENDEd	Read as 1 if usage fault is pended
[11]	SYSTICKACT	Reads as 1 if SysTick is active.

Table 8-21 System Handler Control and State Register bit assignments

Bits	Field	Function
[10]	PENDSVACT	Reads as 1 if PendSV is active.
[9]	-	Reserved
[8]	MONITORACT	Reads as 1 if the Monitor is active.
[7]	SVCALLACT	Reads as 1 if SVCALL is active.
[6:4]	-	Reserved
[3]	USGFAULTACT	Reads as 1 if UsageFault is active.
[2]	-	Reserved
[1]	BUSFAULTACT	Reads as 1 if BusFault is active.
[0]	MEMFAULTACT	Reads as 1 if MemManage is active.

The active bits indicate if any of the system handlers are active, running now, or stacked because of pre-emption. This information is used for debugging and is also used by the application handlers. The pend bits are only set when a fault that cannot be retried has been deferred because of late arrival of a higher priority interrupt.

Caution

You can write, clear, or set the active bits, but you must only do this with extreme caution. Clearing and setting these bits does not repair stack contents nor clean up other data structures. It is intended that context switchers use clearing and setting to save a thread's context, even when in a fault handler. The most common case is to save the context of a thread that is in an SVCALL handler or UsageFault handler, for undefined instruction and coprocessor emulation.

The model for doing this is to save the current state, switch out the stack containing the handler's context, load the state of the new thread, switch in the new thread's stacks, and then return to the thread. The active bit of the current handler must never be cleared, because the IPSR is not changed to reflect this. Only use it to change stacked active handlers.

As indicated, the SVCALLPENDEDED and BUSFAULTPENDEDED bits are set when the corresponding handler is held off by a late arriving interrupt. These bits are not cleared until the underlying handler is actually invoked. That is, if a stack error or vector read error occurs before the SVCALL or BusFault handler is started, the bits are not cleared. This enables the push-error or vector-read-error handler to choose to clear them or retry.

Configurable Fault Status Registers

Use the three Configurable Fault Status Registers to obtain information about local faults. These registers include:

- *Memory Manage Fault Status Register*
- *Bus Fault Status Register* on page 8-34
- *Usage Fault Status Register* on page 8-35.

The flags in these registers indicate the causes of local faults. Multiple flags can be set if more than one fault occurs. These register are read/write-clear. This means that they can be read normally, but writing a 1 to any bit clears that bit.

The register addresses, access types, and Reset states are:

Address	0xE000ED28	Memory Manage Fault Status Register
	0xE000ED29	Bus Fault Status Register
	0xE000ED2A	Usage Fault Status Register
Access	Read/write-one-to-clear	
Reset state	0x00000000	

Figure 8-16 shows the bit assignments of the Configurable Fault Status Registers.

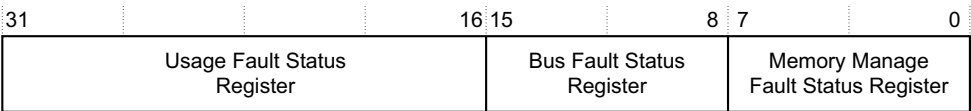


Figure 8-16 Configurable Fault Status Registers bit assignments

Note

Accesses to each individual status register must be aligned to the appropriate address and size. Either the whole 32-bit word is accessed using address 0xE000ED28 or the BFSR and MFSR are accessed as a byte size each correctly aligned or the UFSR is accessed as a half-word correctly aligned to address 0xE000ED2A.

Memory Manage Fault Status Register

The flags in the Memory Manage Fault Status Register indicate the cause of memory access faults.

The register address, access type, and Reset state are:

Address	0xE000ED28
Access	Read/write-one-to-clear
Reset state	0x00000000

Figure 8-17 shows the bit assignments of the Memory Manage Fault Status Register.

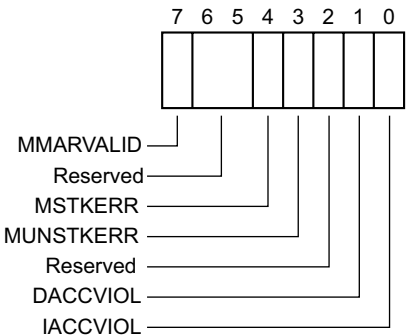


Figure 8-17 Memory Manage Fault Status Register bit assignments

Table 8-22 describes the bit assignments of the Memory Manage Fault Status Register.

Table 8-22 Memory Manage Fault Status Register bit assignments

Bits	Field	Function
[7]	MMARVALID	<i>Memory Manage Address Register (MMAR)</i> address valid flag: 1 = valid fault address in MMAR. A later-arriving fault, such as a bus fault, can clear a memory manage fault. 0 = no valid fault address in MMAR. If a MemManage fault occurs that is escalated to a Hard Fault because of priority, the Hard Fault handler must clear this bit. This prevents problems on return to a stacked active MemManage handler whose MMAR value has been overwritten.
[6:5]	-	Reserved.
[4]	MSTKERR	Stacking from exception has caused one or more access violations. The SP is still adjusted and the values in the context area on the stack might be incorrect. The MMAR is not written.
[3]	MUNSTKERR	Unstack from exception return has caused one or more access violations. This is chained to the handler, so that the original return stack is still present. SP is not adjusted from failing return and new save is not performed. The MMAR is not written.

Table 8-22 Memory Manage Fault Status Register bit assignments (continued)

Bits	Field	Function
[2]	-	Reserved
[1]	DACCVIOL	Data access violation flag. Attempting to load or store at a location that does not permit the operation sets the DACCVIOL flag. The return PC points to the faulting instruction. This error loads MMAR with the address of the attempted access.
[0]	IACCVIOL	Instruction access violation flag. Attempting to fetch an instruction from a location that does not permit execution sets the IACCVIOL flag. This occurs on any access to an XN region, even when the MPU is disabled or not present. The return PC points to the faulting instruction. The MMAR is not written.

Bus Fault Status Register

The flags in the Bus Fault Status Register indicate the cause of bus access faults.

The register address, access type, and Reset state are:

- Address0xE000ED29
- AccessRead/write-one-to-clear
- Reset state0x00000000

Figure 8-18 shows the bit assignments of the Bus Fault Status Register.

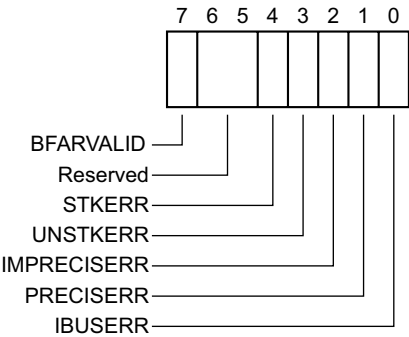


Figure 8-18 Bus Fault Status Register bit assignments

Table 8-23 describes the bit assignments of the Bus Fault Status Register.

Table 8-23 Bus Fault Status Register bit assignments

Bits	Field	Function
[7]	BFARVALID	This bit is set if the <i>Bus Fault Address Register</i> (BFAR) contains a valid address. This is true after a bus fault where the address is known. Other faults can clear this bit, such as a Mem Manage fault occurring later. If a Bus fault occurs that is escalated to a Hard Fault because of priority, the Hard Fault handler must clear this bit. This prevents problems if returning to a stacked active Bus fault handler whose BFAR value has been overwritten.
[6:5]	-	Reserved.
[4]	STKERR	Stacking from exception has caused one or more bus faults. The SP is still adjusted and the values in the context area on the stack might be incorrect. The BFAR is not written.
[3]	UNSTKERR	Unstack from exception return has caused one or more bus faults. This is chained to the handler, so that the original return stack is still present. SP is not adjusted from failing return and new save is not performed. The BFAR is not written.
[2]	IMPRECISERR	Imprecise data bus error. It is a BusFault, but the Return PC is not related to the causing instruction. This is not a synchronous fault. So, if detected when the priority of the current activation is higher than the Bus Fault, it only pends. Bus fault activates when returning to a lower priority activation. If a precise fault occurs before returning to a lower priority exception, the handler detects both IMPRECISERR set and one of the precise fault status bits set at the same time. The BFAR is not written.
[1]	PRECISERR	Precise data bus error return.
[0]	IBUSERR	Instruction bus error flag: 1 = instruction bus error 0 = no instruction bus error. The IBUSERR flag is set by a prefetch error. The fault stops on the instruction, so if the error occurs under a branch shadow, no fault occurs. The BFAR is not written.

Usage Fault Status Register

The flags in the Usage Fault Status Register indicate the following errors:

- illegal combination of EPSR and instruction
- illegal PC load
- illegal processor state
- instruction decode error
- attempt to use a coprocessor instruction
- illegal unaligned access.

The register address, access type, and Reset state are:

Address 0xE000ED2A
Access Read/write clear
Reset state 0x00000000

Figure 8-19 shows the bit assignments of the Usage Fault Status Register.

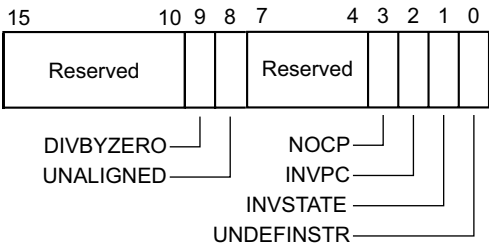


Figure 8-19 Usage Fault Status Register bit assignments

Table 8-24 describes the bit assignments of the Usage Fault Status Register.

Table 8-24 Usage Fault Status Register bit assignments

Bits	Field	Function
[15-10]	-	Reserved
[9]	DIVBYZERO	When DIV_0_TRP (see <i>Configuration Control Register</i> on page 8-26) is enabled and an SDIV or UDIV instruction is used with a divisor of 0, this fault occurs. The instruction is executed and the return PC points to it. If DIV_0_TRP is not set, then the divide returns a quotient of 0.
[8]	UNALIGNED	When UNALIGN_TRP is enabled (see <i>Configuration Control Register</i> on page 8-26), and there is an attempt to make an unaligned memory access, then this fault occurs. Unaligned LDM/STM/LDRD/STRD instructions always fault irrespective of the setting of UNALIGN_TRP.
[7:4]	-	Reserved.
[3]	NOCP	Attempt to use a coprocessor instruction. The processor does not support coprocessor instructions.

Table 8-24 Usage Fault Status Register bit assignments (continued)

Bits	Field	Function
[2]	INVPC	Attempt to load EXC_RETURN into PC illegally. Invalid instruction, invalid context, invalid value. The return PC points to the instruction that tried to set the PC.
[1]	INVSTATE	Invalid combination of EPSR and instruction, for reasons other than UNDEFINED instruction. Return PC points to faulting instruction, with the invalid state.
[0]	UNDEFINSTR	The UNDEFINSTR flag is set when the processor attempts to execute an undefined instruction. This is an instruction that the processor cannot decode. The return PC points to the undefined instruction.

Note

The fault bits are additive if more than one fault occurs before this register is cleared.

Hard Fault Status Register

Use the *Hard Fault Status Register* (HFSR) to obtain information about events that activate the Hard Fault handler.

The register address, access type, and Reset state are:

Address 0xE000ED2C
Access Read/write-one-to-clear
Reset state 0x00000000

The HFSR is a write-clear register. This means that writing a 1 to a bit clears that bit. Figure 8-20 shows the bit assignments of the Hard Fault Status Register.

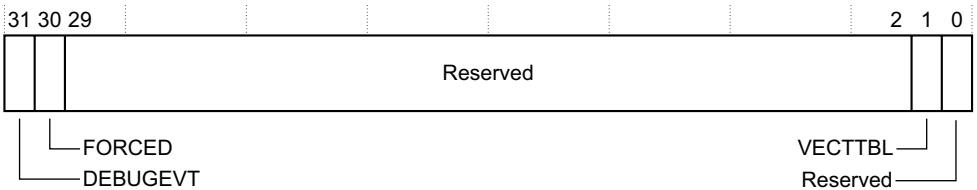


Figure 8-20 Hard Fault Status Register bit assignments

Table 8-25 describes the bit assignments of the Hard Fault Status Register.

Table 8-25 Hard Fault Status Register bit assignments

Bits	Field	Function
[31]	DEBUGEVT	This bit is set if there is a fault related to debug. This is only possible when halting debug is not enabled. For monitor enabled debug, it only happens for BKPT when the current priority is higher than the monitor. When both halting and monitor debug are disabled, it only happens for debug events that are not ignored (minimally, BKPT). The Debug Fault Status Register is updated.
[30]	FORCED	Hard Fault activated because a Configurable Fault was received and cannot activate because of priority or because the Configurable Fault is disabled. The Hard Fault handler then has to read the other fault status registers to determine cause.
[29:2]	-	Reserved.
[1]	VECTTBL	This bit is set if there is a fault because of vector table read on exception processing (Bus Fault). This case is always a Hard Fault. The return PC points to the pre-empted instruction.
[0]	-	Reserved.

Debug Fault Status Register

Use the Debug Fault Status Register to monitor:

- external debug requests
- vector catches
- data watchpoint match
- BKPT instruction execution
- halt requests.

Multiple flags in the Debug Fault Status Register can be set when multiple fault conditions occur. The register is read/write clear. This means that it can be read normally. Writing a 1 to a bit clears that bit.

————— **Note** —————

These bits are not set unless the event is caught. This means that it causes a stop of some sort. If halting debug is enabled, these events stop the processor into debug. If debug is disabled and the debug monitor is enabled, then this becomes a debug monitor handler call, if priority permits. If debug and the monitor are both disabled, some of these events are Hard Faults, and the DBGEVT bit is set in the Hard Fault status register, and some are ignored.

The register address, access type, and Reset state are:

Address 0xE000ED30
Access Read/write-one-to-clear
Reset state 0x00000000

Figure 8-21 shows the bit assignments of the Debug Fault Status Register.

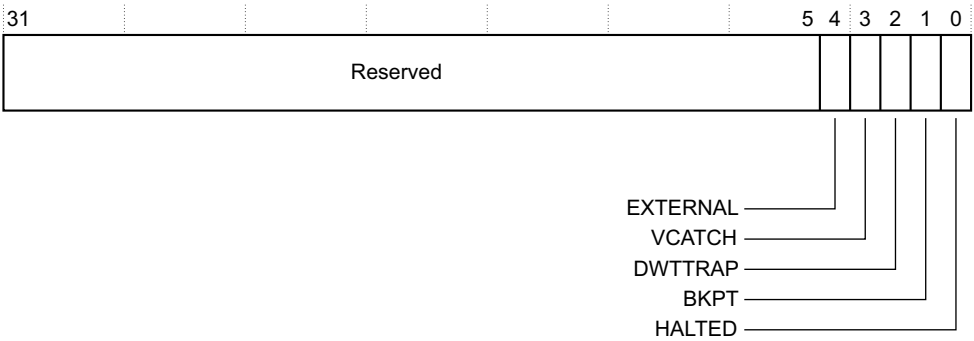


Figure 8-21 Debug Fault Status Register bit assignments

Table 8-26 describes the bit assignments of the Debug Fault Status Register.

Table 8-26 Debug Fault Status Register bit assignments

Bits	Field	Function
[31:5]	-	Reserved
[4]	EXTERNAL	External debug request flag: 1 = EDBGRQ signal asserted 0 = EDBGRQ signal not asserted. The processor stops on next instruction boundary.
[3]	VCATCH	Vector catch flag: 1 = vector catch occurred 0 = no vector catch occurred. When the VCATCH flag is set, a flag in one of the local fault status registers is also set to indicate the type of fault.

Table 8-26 Debug Fault Status Register bit assignments (continued)

Bits	Field	Function
[2]	DWTTRAP	<i>Data Watchpoint and Trace</i> (DWT) flag: 1 = DWT match 0 = no DWT match. The processor stops at the current instruction or at the next instruction.
[1]	BKPT	BKPT flag: 1 = BKPT instruction execution 0 = no BKPT instruction execution. The BKPT flag is set by a BKPT instruction in flash patch code, and also by normal code. Return PC points to breakpoint containing instruction.
[0]	HALTED	Halt request flag: 1 = halt requested by NVIC, including step. The processor is halted on the next instruction. 0 = no halt request.

Memory Manage Fault Address Register

Use the Memory Manage Fault Address Register to read the address of the location that caused a Memory Manage Fault.

The register address, access type, and Reset state are:

Address 0xE000ED34
Access Read/write
Reset state Unpredictable

Table 8-27 describes the field of the Memory Manage Fault Address Register.

Table 8-27 Memory Manage Fault Address Register bit assignments

Bits	Field	Function
[31:0]	ADDRESS	Mem Manage fault address field. ADDRESS is the data address of a faulted load or store attempt. When an unaligned access faults, the address is the actual address that faulted. Because an access can be split into multiple parts, each aligned, this address can be any offset in the range of the requested size. Flags in the Memory Manage Fault Status Register indicate the cause of the fault. See <i>Memory Manage Fault Status Register</i> on page 8-32.

Bus Fault Address Register

Use the Bus Fault Address Register to read the address of the location that generated a Bus Fault.

The register address, access type, and Reset state are:

- Address E000ED38
- Access Read/write
- Reset state Unpredictable

Table 8-28 describes the bit assignments of the Bus Fault Address Register.

Table 8-28 Bus Fault Address Register bit assignments

Bits	Field	Function
[31:0]	ADDRESS	Bus fault address field. ADDRESS is the data address of a faulted load or store attempt. When an unaligned access faults, the address is the address requested by the instruction, even if that is not the address that faulted. Flags in the Bus Fault Status Register indicate the cause of the fault. See <i>Bus Fault Status Register</i> on page 8-34.

Auxiliary Fault Status Register

Use the *Auxiliary Fault Status Register* (AFSR) to determine additional system fault information to software.

The AFSR flags map directly onto the AUXFAULT inputs of the processor, and a single-cycle high level on an external pin causes the corresponding AFSR bit to become latched as one. The bit can only be cleared by writing a one to the corresponding AFSR bit.

When an AFSR bit is written or latched as one, an exception does not occur. If you require an exception, you must use an interrupt.

The register address, access type, and Reset state are:

- Address 0xE000ED3C
- Access Read/write-clear
- Reset state 0x00000000

describes the field of the AFSR.

Table 8-29 Auxiliary Fault Status Register bit assignments

Bits	Field	Function
[31:0]	IMPDEF	Implementation defined. The bits map directly onto the signal assignment to the AUXFAULT inputs. See <i>Miscellaneous</i> on page A-4.

Software Trigger Interrupt Register

Use the Software Trigger Interrupt Register to pend an interrupt to trigger.

The register address, access type, and Reset state are:

Address 0xE000EF00

Access Write-only

Reset state 0x00000000

Figure 8-22 shows the bit assignments of the Software Trigger Interrupt Register.

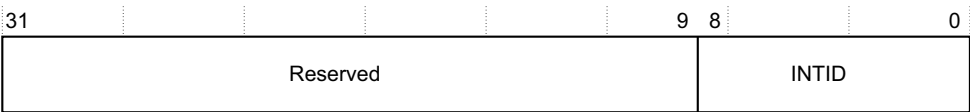


Figure 8-22 Software Trigger Interrupt Register bit assignments

Table 8-30 describes the bit assignments of the Software Trigger Interrupt Register.

Table 8-30 Software Trigger Interrupt Register bit assignments

Bits	Field	Function
[31:9]	-	Reserved.
[8:0]	INTID	Interrupt ID field. Writing a value to the INTID field is the same as manually pending an interrupt by setting the corresponding interrupt bit in an Interrupt Set Pending Register.

8.3 Level versus pulse interrupts

The processor supports both level and pulse interrupts. A level interrupt is held asserted until it is cleared by the ISR accessing the device. A pulse interrupt is a variant of an edge model. The edge must be sampled on the rising edge of the Cortex-M3 clock, **FCLK**, instead of being asynchronous.

For level interrupts, if the signal is not deasserted before the return from the interrupt routine, the interrupt repends and re-activates. This is particularly useful for FIFO and buffer-based devices because it ensures that they drain either by a single ISR or by repeated invocations, with no extra work. This means that the device holds the signal in assert until the device is empty.

A pulse interrupt can be reasserted during the ISR so that the interrupt can be pended and active at the same time. The application design must ensure that a second pulse does not arrive before the first pulse is activated. The second pend has no affect because it is already pended. However, if the interrupt is asserted for at least one cycle, the NVIC latches the pend bit. When the ISR activates, the pend bit is cleared. If the interrupt asserts again while it is activated, it can latch the pend bit again.

Pulse interrupts are mostly used for external signals and for rate or repeat signals.

Chapter 9

Memory Protection Unit

This chapter describes the processor *Memory Protection Unit* (MPU). It contains the following sections:

- *About the MPU* on page 9-2
- *MPU programmer's model* on page 9-3
- *Interrupts and updating the MPU* on page 9-19
- *MPU access permissions* on page 9-13
- *MPU aborts* on page 9-15
- *Updating an MPU region* on page 9-16.

9.1 About the MPU

The MPU is an optional component for memory protection. The processor supports the standard ARMv7 *Protected Memory System Architecture* (PMSAv7) model. The MPU provides full support for:

- protection regions
- overlapping protection regions, with ascending region priority:
 - 7 = highest priority
 - 0 = lowest priority.
- access permissions
- exporting memory attributes to the system.

MPU mismatches and permission violations invoke the programmable-priority MemManage fault handler. For more information, see *Memory Manage Fault Address Register* on page 8-40.

You can use the MPU to:

- enforce privilege rules
- separate processes
- enforce access rules.

9.2 MPU programmer's model

This sections describes the registers that control the MPU. It contains the following:

- *Summary of the MPU registers*
- *Description of the MPU registers.*

9.2.1 Summary of the MPU registers

Table 9-1 provides a summary of the MPU registers.

Table 9-1 MPU registers

Name of register	Type	Address	Reset value	Page
MPU Type Register	Read Only	0xE000ED90	0x00000800	page 9-3
MPU Control Register	Read/Write	0xE000ED94	0x00000000	page 9-4
MPU Region Number register	Read/Write	0xE000ED98	-	page 9-6
MPU Region Base Address register	Read/Write	0xE000ED9C	-	page 9-7
MPU Region Attribute and Size register(s)	Read/Write	0xE000EDA0	-	page 9-8
MPU Alias 1 Region Base Address register	Alias of D9C	0xE000EDA4	-	page 9-11
MPU Alias 1 Region Attribute and Size register	Alias of DA0	0xE000EDA8	-	page 9-11
MPU Alias 2 Region Base Address register	Alias of D9C	0xE000EDAC	-	page 9-11
MPU Alias 2 Region Attribute and Size register	Alias of DA0	0xE000EDB0	-	page 9-11
MPU Alias 3 Region Base Address register	Alias of D9C	0xE000EDB4	-	page 9-11
MPU Alias 3 Region Attribute and Size register	Alias of DA0	0xE000EDB8	-	page 9-11

9.2.2 Description of the MPU registers

This section contains a description of the MPU registers.

MPU Type Register

Use the MPU Type Register to see how many regions the MPU supports. Read bits [15:8] to determine if an MPU is present.

The register address, access type, and Reset state are:

Address 0xE000ED90

Access Read-only

Reset state 0x00000800

Figure 9-1 shows the bit assignments of the MPU Type Register.

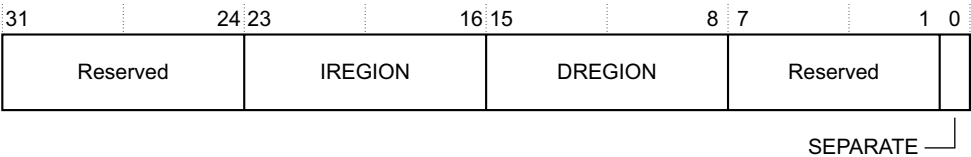


Figure 9-1 MPU Type Register bit assignments

Table 9-2 describes the bit assignments of the MPU Type Register.

Table 9-2 MPU Type Register bit assignments

Bits	Field	Function
[31:24]	-	Reserved.
[23:16]	IREGION	Because the processor core uses only a unified MPU, IREGION always contains 0x00.
[15:8]	DREGION	Number of supported MPU regions field. DREGION contains 0x08 if the implementation contains an MPU indicating eight MPU regions, otherwise it contains 0x00.
[7:0]	-	Reserved.
[0]	SEPARATE	Because the processor core uses only a unified MPU, SEPARATE is always 0.

MPU Control Register

Use the MPU Control Register to:

- enable the MPU
- enable the default memory map (background region)
- enable the MPU when in Hard Fault, *Non-maskable Interrupt* (NMI), and FAULTMASK escalated handlers.

When the MPU is enabled, at least one region of the memory map must be enabled for the MPU to function unless the PRIVDEFENA bit is set. If the PRIVDEFENA bit is set and no regions are enabled, then only privileged code can operate.

When the MPU is disabled, the default address map is used, as if no MPU is present.

When the MPU is enabled, only the system partition and vector table loads are always accessible. Other areas are accessible based on regions and whether PRIVDEFENA is enabled.

Unless HFNMIENA is set, the MPU is not enabled when the exception priority is –1 or –2. These priorities are only possible when in Hard fault, NMI, or when FAULTMASK is enabled. The HFNMIENA bit enables the MPU when operating with these two priorities.

The register address, access type, and Reset state are:

Address 0xE000ED94

Access Read/write

Reset state 0x00000000

Figure 9-2 shows the bit assignments of the MPU Control Register.

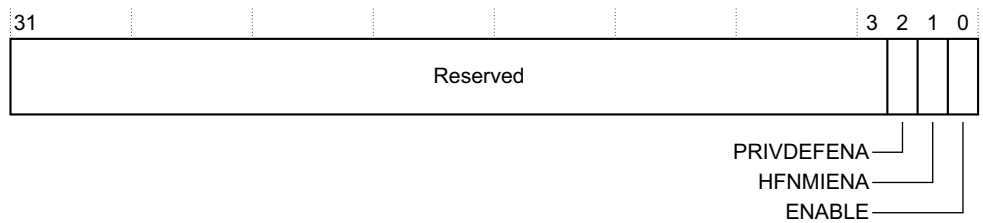


Figure 9-2 MPU Control Register bit assignments

Table 9-3 describes the bit assignments of the MPU Control Register.

Table 9-3 MPU Control Register bit assignments

Bits	Field	Function
[31:2]	-	Reserved.
[2]	PRIVDEFENA	<p>This bit enables the default memory map for privileged access, as a background region, when the MPU is enabled. The background region acts as if it was region number 1 before any settable regions. Any region that is set up overlays this default map, and overrides it.</p> <p>If this bit = 0, the default memory map is disabled, and memory not covered by a region faults.</p> <p>When the MPU is enabled and PRIVDEFENA is enabled, the default memory map is as described in Chapter 4 <i>Memory Map</i>. This applies to memory type, <i>Execute Never</i> (XN), cache and shareable rules. However, this only applies to privileged mode (fetch and data access). User mode code faults unless a region has been set up for its code and data.</p> <p>When the MPU is disabled, the default map acts on both privileged and user mode code. XN and SO rules always apply to the System partition whether this enable is set or not.</p> <p>If the MPU is disabled, this bit is ignored.</p> <p>Reset clears the PRIVDEFENA bit.</p>
[1]	HFNMIENA	<p>This bit enables the MPU when in Hard Fault, NMI, and FAULTMASK escalated handlers. If this bit = 1 and the ENABLE bit = 1, the MPU is enabled when in these handlers. If this bit = 0, the MPU is disabled when in these handlers, regardless of the value of ENABLE. If this bit =1 and ENABLE = 0, behavior is Unpredictable.</p> <p>Reset clears the HFNMIENA bit.</p>
[0]	ENABLE	<p>MPU enable bit:</p> <p>1 = enable MPU</p> <p>0 = disable MPU.</p> <p>Reset clears the ENABLE bit.</p>

MPU Region Number Register

Use the MPU Region Number Register to select which protection region is accessed. Then write to the MPU Region Base Address Register or the MPU Attributes and Size Register to configure the characteristics of the protection region.

The register address, access type, and Reset state are:

- Address 0xE000ED98
- Access Read/write
- Reset state Unpredictable

Figure 9-3 on page 9-7 shows the bit assignments of the MPU Region Number Register.

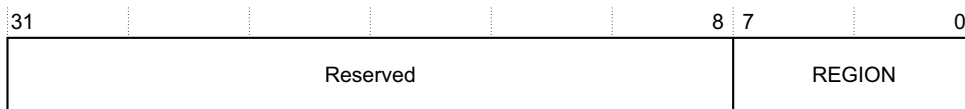
**Figure 9-3 MPU Region Number Register bit assignments**

Table 9-4 describes the bit assignments of the MPU Region Number Register.

Table 9-4 MPU Region Number Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	REGION	Region select field. Selects the region to operate on when using the Region Attribute and Size Register and the Region Base Address Register. It must be written first except when the address VALID + REGION fields are written, which overwrites this.

MPU Region Base Address Register

Use the MPU Region Base Address Register to write the base address of a region. The Region Base Address Register also contains a REGION field that you can use to override the REGION field in the MPU Region Number Register, if the VALID bit is set.

The Region Base Address register sets the base for the region. It is aligned by the size. So, a 64-KB sized region must be aligned on a multiple of 64KB, for example, 0x00010000 or 0x00020000.

The region always reads back as the current MPU region number. VALID always reads back as 0. Writing with VALID = 1 and REGION = n changes the region number to n. This is a short-hand way to write the MPU Region Number Register.

This register is Unpredictable if accessed other than as a word.

The register address, access type, and Reset state are:

Address 0xE00ED9C
Access Read/write
Reset state Unpredictable

Figure 9-4 on page 9-8 shows the bit assignments of the MPU Region Base Address Register.

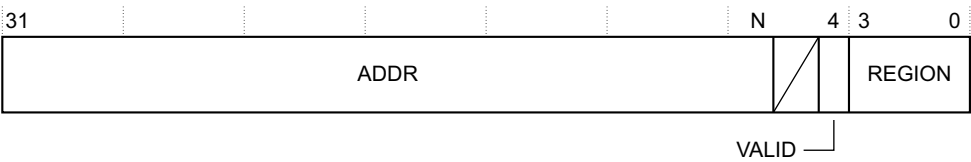


Figure 9-4 MPU Region Base Address Register bit assignments

Table 9-5 describes the bit assignments of the MPU Region Base Address Register.

Table 9-5 MPU Region Base Address Register bit assignments

Bits	Field	Function
[31:N]	ADDR	Region base address field. The value of N depends on the region size, so that the base address is aligned according to an even multiple of size. The power of 2 size specified by the SZENABLE field of the MPU Region Attribute and Size Register defines how many bits of base address are used.
[4]	VALID	MPU Region Number valid bit: 1 = MPU Region Number Register is overwritten by bits 3:0 (the REGION value). 0 = MPU Region Number Register remains unchanged and is interpreted.
[3:0]	REGION	MPU region override field.

MPU Region Attribute and Size Register

Use the MPU Region Attribute and Size Register to control the MPU access permissions. The register is made up of two part registers, each of halfword size. These can be accessed using the individual size, or they can both be simultaneously accessed using a word operation.

The sub-region disable bits are not supported for region sizes of 32 bytes, 64 bytes, and 128 bytes. When these region sizes are used, the subregion disable bits must be programmed as 0.

The register address, access type, and Reset state are:

- Address 0xE000EDA0
- Access Read/write
- Reset state Unpredictable

Figure 9-5 on page 9-9 shows the bit assignments of the MPU Region Attribute and Size Register.

31	29	28	27	26	24	23	22	21	19	18	17	16	15			8	7	6	5			1	0
Res	XN	Res	AP	Res.	TEX	S	C	B	SRD						Res.	SIZE				EN A			

Figure 9-5 MPU Region Attribute and Size Register bit assignments

Table 9-6 describes the bit assignments of the MPU Region Attribute and Size Register. For more information, see *MPU access permissions* on page 9-13.

Table 9-6 MPU Region Attribute and Size Register bit assignments

Bits	Field	Function	
[31:29]	-	Reserved.	
[28]	XN	Instruction access disable bit: 1 = disable instruction fetches 0 = enable instruction fetches.	
[27]	-	Reserved.	
[26:24]	AP	Data access permission field:	
	Value	Privileged permissions	User permissions
	b000	No access	No access
	b001	Read/write	No access
	b010	Read/write	Read-only
	b011	Read/write	Read/write
	b100	Reserved	Reserved
	b101	Read-only	No access
	b110	Read-only	Read-only
	b111	Read-only	Read-only.
[23:22]	-	Reserved.	
[21:19]	TEX	Type extension field.	
[18]	S	Shareable bit: 1 = shareable 0 = not shareable.	
[17]	C	Cacheable bit: 1 = cacheable 0 = not cacheable.	

Table 9-6 MPU Region Attribute and Size Register bit assignments (continued)

Bits	Field	Function
[16]	B	Bufferable bit: 1 = bufferable 0 = not bufferable.
[15:8]	SRD	<i>Sub-Region Disable</i> (SRD) field. Setting an SRD bit disables the corresponding sub-region. Regions are split into eight equal-sized sub-regions. Sub-regions are not supported for region sizes of 128 bytes and less. For more information, see <i>Sub-Regions</i> on page 9-12.
[7:6]	-	Reserved.
[5:1]	SIZE	MPU Protection Region Size Field. See Table 9-7.
[0]	ENABLE	Region enable bit.

For information about access permission, see *MPU access permissions* on page 9-13.

Table 9-7 MPU protection region size field

Region	Size
b00000	Reserved
b00001	Reserved
b00010	Reserved
b00011	Reserved
b00100	32B
b00101	64B
b00110	128B
b00111	256B
b01000	512B
b01001	1KB
b01010	2KB
b01011	4KB
b01100	8KB
b01101	16KB

Table 9-7 MPU protection region size field (continued)

Region	Size
b01110	32KB
b01111	64KB
b10000	128KB
b10001	256KB
b10010	512KB
b10011	1MB
b10100	2MB
b10101	4MB
b10110	8MB
b10111	16MB
b11000	32MB
b11001	64MB
b11010	128MB
b11011	256MB
b11100	512MB
b11101	1GB
b11110	2GB
b11111	4GB

9.2.3 Accessing the MPU using the alias registers

You can optimize the loading speed of the MPU registers using register aliasing. There are three sets of *Nested Vectored Interrupt Controller* (NVIC) alias registers. These are described in *NVIC register descriptions* on page 8-7.

The aliases access the registers in exactly the same way, and they exist to enable the use of sequential writes (STM) to update between one and four regions. This is used when disable/change/enable is not required.

You cannot use these aliases to read the contents of the regions because the region number must be written.

An example code sequence for updating four regions is

```
; R1 = 4 region pairs from process control block (8 words)
MOV R0, #NVIC_BASE
ADD R0, #MPU_REG_CTRL
LDM R1, [R2-R9] ; load region information for 4 regions
STM R0, [R2-R9] ; update all 4 regions at once
```

Note

You can normally use the `memcpy()` function in a C/C++ compiler for this sequence. However, you must verify that the compiler uses word transfers.

9.2.4 Sub-Regions

The eight *Sub-Region Disable* (SRD) bits of the Region Attribute and Size Register divide a region into eight equal-sized units based on the region size. This enables selectively disabling some of the 1/8th sub-regions. The least significant bit affects the first 1/8th sub-region, and the most significant bits affects the last 1/8th sub-region. A disabled sub-region enables any other region overlapping that range to be matched instead. If no other region overlaps the sub-region, the default behavior is used, no match – a fault. Sub-regions cannot be used with the three smallest regions of size: 32, 64, and 128. If these sub-regions are used, the results are Unpredictable.

Example of SRD use

Two regions with the same base address overlap. One region is 64KB, and the other is 512KB. The bottom 64KB of the 512KB region is disabled so that the attributes from the 64KB apply. This is achieved by setting SRD for the 512KB region to b00000001.

9.3 MPU access permissions

This section describes the MPU access permissions. The access permission bits, TEX, C, B, AP, and XN, of the Region Access Control Register (see *MPU Region Attribute and Size Register* on page 9-8) control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then a permission fault is raised.

Table 9-8 describes the TEX, C, and B encoding.

Table 9-8 TEX, C, B encoding

TEX	C	B	Description	Memory type	Region shareability
b000	0	0	Strongly ordered.	Strongly ordered	Shareable
b000	0	1	Shared device.	Device	Shareable
b000	1	0	Outer and inner write-through. No write allocate.	Normal	S
b000	1	1	Outer and inner write-back. No write allocate.	Normal	S
b001	0	0	Outer and inner noncacheable.	Normal	S
b001	0	1	Reserved.	Reserved	Reserved
b001	1	0	Implementation-defined.		
b001	1	1	Outer and inner write-back. Write and read allocate.	Normal	S
b010	0	0	Nonshared device.	Device	Not shareable
b010	0	1	Reserved.	Reserved	Reserved
b010	1	X	Reserved.	Reserved	Reserved
b1BB	A	A	Cached memory BB = outer policy. AA = inner policy.	Normal	S

Note

In Table 9-8, S is the S bit [2] from the MPU Region Attributes and Size Register.

Table 9-9 describes the cache policy for memory attribute encoding.

Table 9-9 Cache policy for memory attribute encoding

Memory attribute encoding (AA and BB)	Cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

Note

All cache policies presented by **HPROT** and **MEMATTR** relate to an outer cache.

Table 9-10 describes the AP encoding.

Table 9-10 AP encoding

AP[2:0]	Privileged permissions	User permissions	Descriptions
000	No access	No access	All accesses generate a permission fault
001	Read/write	No access	Privileged access only
010	Read/write	Read only	Writes in user mode generate a permission fault
011	Read/write	Read/write	Full access
100	Unpredictable	Unpredictable	Reserved
101	Read only	No access	Privileged read only
110	Read only	Read only	Privileged/user read only
111	Read only	Read only	Privileged/user read only

Table 9-11 describes the XN encoding.

Table 9-11 XN encoding

XN	Description
0	All instruction fetches enabled
1	Instruction fetches disabled

9.4 MPU aborts

For information about MPU aborts, see *Memory Manage Fault Address Register* on page 8-40.

9.5 Updating an MPU region

There are three registers consisting of three memory mapped words that program the MPU regions. These are part registers that you can individually program and access. This means that you can port existing ARMv6, ARMv7, and CP15 code. This replaces MRC and MCR with LDRx and STRx operations.

You can also access these registers as three words, and program them using only two words. Aliases are provided to enable programming a set of regions simultaneously using an STM instruction.

9.5.1 Updating an MPU region using CP15 equivalent code

Using CP15 equivalent code:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
STR R4,[R0,#4]; address
STRH R2,[R0,#8]; size and enable
STRH R3,[R0,#10]; attributes
```

Note

If interrupts could pre-empt during this period, this region could affect them. This means that the region must be disabled, written, and then enabled. This is usually not necessary for a context switcher, but would be necessary if updated elsewhere.

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
BIC R2,R2, #1; disable
STRH R2,[R0,#8]; size and enable
STR R4,[R0,#4]; address
STRH R3,[R0,#10]; attributes
ORR R2,#1; enable
STRH R2,[R0,#8]; size and enable
```

DMB/DSB is not necessary because the Private Peripheral Bus is a strongly ordered memory area. However, a DSB is necessary before the effect on the MPU takes place, such as the end of a context switcher.

An ISB is necessary if the code that programs the MPU region or regions is entered using a branch or call. If the code is entered using a return from exception, or by taking an exception, then an ISB is not necessary.

9.5.2 Updating an MPU region using two or three words

You can program directly using two or three words, depending on how the information is divided:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
STR R2,[R0,#4]; address
STR R3,[R0,#8]; size, attributes
```

An STM can optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STM R0,{R1-R3}; region number, address, size, and attributes
```

You can do this in two words for pre-packed information. This means that the base address register contains the region number in addition to a region-valid bit. This is useful when the data is statically packed, for example in a boot list or a *Process Control Block* (PCB).

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#4]; address and region number
STR R2,[R0,#8]; size and attributes
```

An STM can optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
```

STM R0,{R1-R2}; address, region number, size

For information about interrupts and updating the MPU, see *Interrupts and updating the MPU* on page 9-19.

9.6 Interrupts and updating the MPU

An MPU region can contain critical data. This is because it takes more than one bus transaction to update. This is normally two words. As a result, it is not thread safe. That is, an interrupt can split the two words, leaving the region with incoherent information. There are two different issues:

- An interrupt can come in that would also update the MPU. This is not only a read-modify-write issue, it also affects cases where the interrupt routine is guaranteed not to modify the same region. This is because the programming relies on the region number being written into a register so that it knows which region to update. So in this case, you must disable interrupts around each update routine.
- An interrupt can come in that would use the region being updated or would be affected because only the base or size fields had been updated. If the new size field is changed, but the base is not, the base+new_size might overlap into an area normally handled by another region. In this case, the disable-modify-enable approach is required.

But for standard OS context switch code, which would change user regions, there is no risk, because these regions would be preset to user privilege and a user area address. This means that even an interrupt would cause no side effect. Therefore the disable/enable code is not required nor is interrupt disable.

The most common approach is to only program the MPU from boot code and context switcher. If these are the only two places, and the context switcher is only updating user regions, then disable is not required because the context switcher is already a critical region and the boot code runs with interrupts disabled.

Chapter 10

Core Debug

This chapter describes how to debug and test the processor. It contains the following sections:

- *About core debug* on page 10-2
- *Core debug registers* on page 10-3
- *Core debug access example* on page 10-12
- *Using application registers in core debug* on page 10-13.

10.1 About core debug

Core debug is accessed through the core debug registers. Debug access to these registers is by means of the *Advanced High-performance Bus* (AHB-AP) port, see *AHB-AP* on page 11-39. The processor can access these registers directly over the internal *Private Peripheral Bus* (PPB).

Table 10-1 shows the core debug registers.

Table 10-1 Core debug registers

Address	Type	Reset Value	Description
0xE000EDF0	Read/Write	0x00000000 ^a	Debug Halting Control and Status Register
0xE000EDF4	Write-only	-	Debug Core Register Selector Register
0xE000EDF8	Read/Write	-	Debug Core Register Data Register
0xE000EDFC	Read/Write	0x00000000 ^b	Debug Exception and Monitor Control Register.

- a. Bits 5, 3, 2, 1, 0 are reset by **PORESETn**. Bit [1] is also reset by **SYSRESETn** and writing a 1 to the **VECTRESET** bit of the Application Interrupt and Reset Control Register.
- b. Bits 16,17,18,19 are also reset by **SYSRESETn** and writing a 1 to the **VECTRESET** bit of the Application Interrupt and Reset Control Register.

The Debug Fault Status Register is also used for debug purposes. See *Debug Fault Status Register* on page 8-38 for more information.

Core debug is an optional component. If core debug is removed then halt mode debugging is not supported, therefore there is no halt, stepping, or register transfer functionality. Debug monitor mode is still supported.

10.1.1 Halt mode debugging

The debugger can halt the core by setting the **C_DEBUGEN** and **C_HALT** bits of the Debug Halting Control and Status Register. The core acknowledges when halted by setting the **S_HALT** bit of the Debug Halting Control and Status Register.

The core can be single stepped by halting the core, setting the **C_STEP** bit to 1, and then clearing the **C_HALT** bit to 0. The core acknowledges completion of the step and re-halt by setting the **S_HALT** bit of the Debug Halting Control and Status Register.

10.1.2 Exiting core debug

The core can exit Halting debug by clearing the **C_DEBUGEN** bit in the Debug Halting and Status Register.

10.2 Core debug registers

The registers that provide debug operations are:

- *Debug Halting Control and Status Register*
- *Debug Exception and Monitor Control Register* on page 10-8.
- *Debug Core Register Data Register* on page 10-8
- *Debug Exception and Monitor Control Register* on page 10-8.

10.2.1 Debug Halting Control and Status Register

The purpose of the *Debug Halting Control and Status Register* (DHCSR) is to:

- provide status information about the state of the processor
- enable core debug
- halt and step the processor.

The DHCSR:

- is a 32-bit read/write register
- address is 0xE000EDF0.

Note

The DHCSR is only reset from a system reset, including power on. Bit 16 of DHCSR is Unpredictable on reset.

Figure 10-1 on page 10-4 shows the bit assignments in the register.

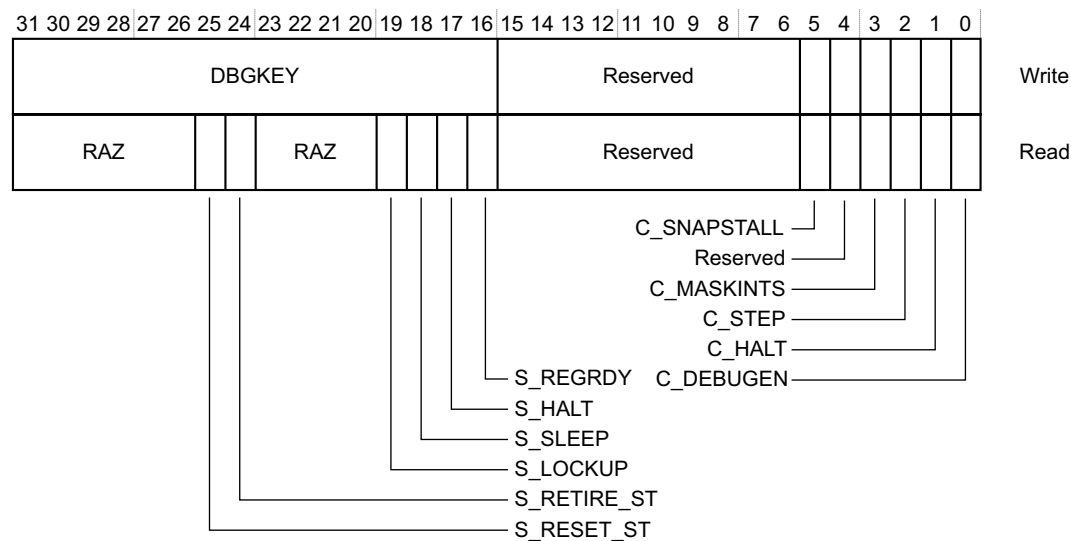


Figure 10-1 Debug Halting Control and Status Register bit assignments

Table 10-2 shows the bit functions of the Debug ID Register.

Table 10-2 Debug Halting Control and Status Register

Bits	Type	Field	Function
[31:16]	Write	DBGKEY	Debug Key. 0xA05F must be written whenever this register is written. Reads back as status bits [25:16]. If not written as Key, the write operation is ignored and no bits are written into the register.
[31:26]	-	-	Reserved, RAZ.
[25]	Read	S_RESET_ST	Indicates that the core has been reset, or is now being reset, since the last time this bit was read. This a sticky bit that clears on read. So, reading twice and getting 1 then 0 means it was reset in the past. Reading twice and getting 1 both times means that it is being reset now (held in reset still).
[24]	Read	S_RETIRE_ST	Indicates that an instruction has completed since last read. This is a sticky bit that clears on read. This determines if the core is stalled on a load/store or fetch.
[23:20]	-	-	Reserved, RAZ.
[19]	Read	S_LOCKUP	Reads as one if the core is running (not halted) and a lockup condition is present.

Table 10-2 Debug Halting Control and Status Register (continued)

Bits	Type	Field	Function
[18]	Read	S_SLEEP	Indicates that the core is sleeping (WFI, WFE, or SLEEP-ON-EXIT). Must use C_HALT to gain control or wait for interrupt to wake-up. For more information on SLEEP-ON-EXIT see Table 7-1 on page 7-3.
[17]	Read	S_HALT	The core is in debug state when S_HALT is set.
[16]	Read	S_REGRDY	Register Read/Write on the Debug Core Register Selector register is available. Last transfer is complete.
[15:6]	-	-	Reserved.
[5]	Read/write	C_SNAPSTALL	<p>If the core is stalled on a load/store operation the stall ceases and the instruction is forced to complete. This enables Halting debug to gain control of the core. It can only be set if:</p> <p>C_DEBUGEN = 1</p> <p>C_HALT = 1.</p> <p>The core reads S_RETIRE_ST as 0. This indicates that no instruction has advanced. This prevents misuse.</p> <p>The bus state is Unpredictable when this is used.</p> <p>S_RETIRE can detect core stalls on load/store operations.</p>
[4]	-	-	Reserved.
[3]	Read/write	C_MASKINTS	Mask interrupts when stepping or running in halted debug. Does not affect NMI, which is not maskable. Must only be modified when the processor is halted (S_HALT == 1). Also does not affect fault exceptions and SVC caused by execution of the instructions. CMASKINTS must be set or cleared before halt is released. This means that the writes to set or clear C_MASKINTS and to set or clear C_HALT must be separate.
[2]	Read/write	C_STEP	Steps the core in halted debug. When C_DEBUGEN = 0, this bit has no effect. Must only be modified when the processor is halted (S_HALT == 1).
[1]	Read/write	C_HALT	Halts the core. This bit is set automatically when the core Halts. For example Breakpoint. This bit clears on core reset. This bit can only be written if C_DEBUGEN is 1, otherwise it is ignored. When setting this bit to 1, C_DEBUGEN must also be written to 1 in the same value (value[1:0] is 2'b11). The core can halt itself, but only if C_DEBUGEN is already 1 and only if it writes with b11).
[0]	Read/write	C_DEBUGEN	<p>Enables debug. This can only be written by AHB-AP and not by the core. It is ignored when written by the core, which cannot set or clear it.</p> <p>The core must write a 1 to it when writing C_HALT to halt itself.</p>

If not enabled for Halting mode, **C_DEBUGEN** = 1, all other fields are disabled.

This register is not reset on a system reset. It is reset by a power-on reset. However, the **C_HALT** bit always clears on a system reset.

To halt on a reset, the following bits must be enabled:

- bit [0], **VC_CORERESET**, of the Debug Exception and Monitor Control Register
- bit [0], **C_DEBUGEN**, of the Debug Halting Control and Status Register.

————— Note —————

Writes to this register in any size other than word are Unpredictable. It is acceptable to read in any size, and you can use it to avoid or intentionally change a sticky bit.

10.2.2 Debug Core Register Selector Register

The purpose of the *Debug Core Register Selector Register* (DCRSR) is to select the processor register to transfer data to or from.

The DCRSR:

- is a 17-bit write-only register
- address is 0xE000EDF4.

Figure 10-2 shows the bit assignments in the register.

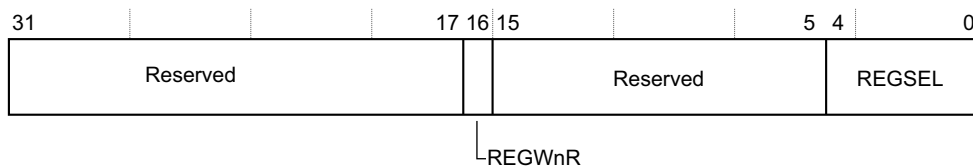


Figure 10-2 Debug Core Register Selector Register bit assignments

Table 10-3 shows the bit functions of the Debug Core Selector Register.

Table 10-3 Debug Core Register Selector Register

Bits	Type	Field	Function
[31:17]	-	-	Reserved
[16]	Write	REGWnR	Write = 1 Read = 0
[15:5]	-	-	Reserved.
[4:0]	Write	REGSEL	5b00000 = R0 5b00001 = R1 ... 5b01111 = DebugReturnAddress() 5b10000 = xPSR/Flags, execution state information, and exception number 5b10001 = <i>MSP</i> (Main SP) 5b10010 = <i>PSP</i> (Process SP) 5b10100: <ul style="list-style-type: none">CONTROL bits [31:24]FAULTMASK bits [23:16]BASEPRI bits [15:8]PRIMASK bits [7:0] All unused values reserved

This write-only register generates a handshake to the core to transfer data to or from Debug Core Register Data Register and the selected register. Until this core transaction is complete, bit [16], **S_REGRDY**, of the DHCSR is 0.

Note

- Writes to this register in any size but word are Unpredictable.
- PSR registers are fully accessible this way, whereas some read as 0 when using MRS instructions.
- All bits can be written, but some combinations cause a fault when execution is resumed.
- IT might be written and behaves as though in an IT block.

- ICI can be written, though invalid values or when not used on an LDM/STM causes a fault, as would on return from exception. Changing ICI from a value to 0 causes the underlying LDM/STM to start, not continue.
-

10.2.3 Debug Core Register Data Register

The purpose of the *Debug Core Register Data Register* (DCRDR) is to hold data for reading and writing registers to and from the processor.

The DCRDR:

- is a 32-bit read/write register
- address 0xE000EDF8.

This is the data value written to the register selected by the Debug Register Selector Register.

When the processor receives a request from the Debug Core Register Selector, this register is read or written by the processor using a normal load-store unit operation.

If core register transfers are not being performed, software-based debug monitors can use this register for communication in non-halting debug. For example, OS RSD and Real View Monitor. This enables flags and bits to acknowledge state and indicate if commands have been accepted to, replied to, or accepted and replied to.

10.2.4 Debug Exception and Monitor Control Register

The purpose of the *Debug Exception and Monitor Control Register* (DEMCR) is:

- Vector catching. That is, to cause debug entry when a specified vector is committed for execution.
- Debug monitor control.

The DEMCR:

- is a 32-bit read/write register
- has address 0xE000EDFC.

Figure 10-2 on page 10-6 shows the bit assignments in the register.

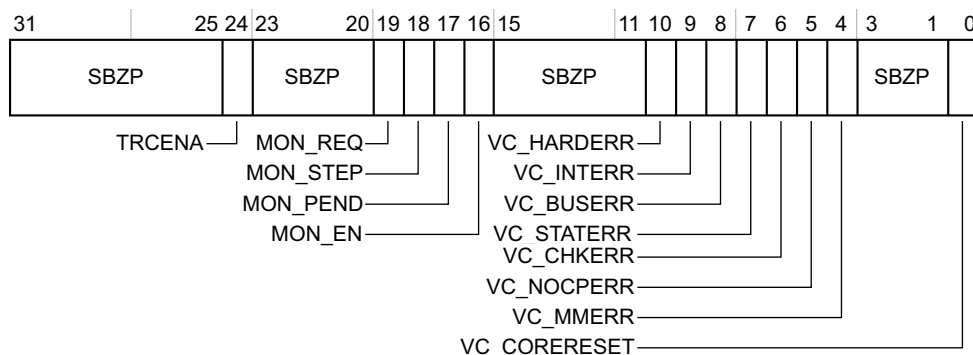


Figure 10-3 Debug Exception and Monitor Control Register bit assignments

Table 10-4 shows the bit functions of the Debug Exception and Monitor Control Register.

Table 10-4 Debug Exception and Monitor Control Register

Bits	Type	Field	Function
[31:25]	-	-	Reserved, SBZP
[24]	Read/write	TRCENA	<p>This bit must be set to 1 to enable use of the trace and debug blocks:</p> <ul style="list-style-type: none"> • <i>Data Watchpoint and Trace (DWT)</i> • <i>Instrumentation Trace Macrocell (ITM)</i> • <i>Embedded Trace Macrocell (ETM)</i> • <i>Trace Port Interface Unit (TPIU).</i> <p>This enables control of power usage unless tracing is required. The application can enable this, for ITM use, or use by a debugger.</p> <p>———— Note ————</p> <p>If no debug or trace components are present in the implementation then it is not possible to set TRCENA.</p>
[23:20]	-	-	Reserved, SBZP
[19]	Read/write	MON_REQ^a	<p>This enables the monitor to identify how it wakes up:</p> <p>1 = woken up by MON_PEND</p> <p>0 = woken up by debug exception.</p>
[18]	Read/write	MON_STEP^a	<p>When MON_EN = 1, this steps the core. When MON_EN = 0, this bit is ignored. This is the equivalent to C_STEP. Interrupts are only stepped according to the priority of the monitor and settings of PRIMASK, FAULTMASK, or BASEPRI.</p>

Table 10-4 Debug Exception and Monitor Control Register (continued)

Bits	Type	Field	Function
[17]	Read/write	MON_PEND^a	<p>Pend the monitor to activate when priority permits. This can wake up the monitor through the AHB-AP port. It is the equivalent to C_HALT for Monitor debug.</p> <p>This register does not reset on a system reset. It is only reset by a power-on reset. Software in the reset handler or later, or by the DAP must enable the debug monitor.</p>
[16]	Read/write	MON_EN^a	<p>Enable the debug monitor. When enabled, the System handler priority register controls its priority level. If disabled, then all debug events go to Hard fault. C_DEBUGEN in the Debug Halting Control and Statue register overrides this bit.</p> <p>Vector catching is semi-synchronous. When a matching event is seen, a Halt is requested. Because the processor can only halt on an instruction boundary, it must wait until the next instruction boundary. As a result, it stops on the first instruction of the exception handler. However, two special cases exist when a vector catch has triggered:</p> <ul style="list-style-type: none"> • If a fault is taken during vectoring, vector read or stack push error, the halt occurs on the corresponding fault handler, for the vector error or stack push. • If a late arriving interrupt comes in during vectoring, it is not taken. That is, an implementation that supports the late arrival optimization must suppress it in this case.
[15:11]	-	-	Reserved, SBZP
[10]	Read/write	VC_HARDERR^b	Debug trap on Hard Fault.
[9]	Read/write	VC_INTERR^b	Debug Trap on interrupt/exception service errors. These are a subset of other faults and catches before BUSERR or HARDERR.
[8]	Read/write	VC_BUSERR^b	Debug Trap on normal Bus error.
[7]	Read/write	VC_STATERR^b	Debug trap on Usage Fault state errors.
[6]	Read/write	VC_CHKERR^b	Debug trap on Usage Fault enabled checking errors.
[5]	Read/write	VC_NOCPERR^b	Debug trap on Usage Fault access to Coprocessor that is not present or marked as not present in CAR register.
[4]	Read/write	VC_MMERR^b	Debug trap on Memory Management faults.
[3:1]	-	-	Reserved, SBZP
[0]	Read/write	VC_CORERESET^b	Reset Vector Catch. Halt running system if Core reset occurs.

- a. This bit clears on a Core Reset.
- b. Only usable when **C_DEBUGEN** = 1.

This register manages exception behavior under debug.

Vector catching is only available to halting debug. The upper halfword is for monitor controls and the lower halfword is for halting exception support.

This register is not reset on a system reset.

This register is reset by a power-on reset. Bits [19:16] are always cleared on a core reset. The debug monitor is enabled by software in the reset handler or later, or by the AHB-AP port.

Vector catching is semi-synchronous. When a matching event is seen, a Halt is requested. Because the processor can only halt on an instruction boundary, it must wait until the next instruction boundary. As a result, it stops on the first instruction of the exception handler. However, two special cases exist when a vector catch has triggered:

1. If a fault is taken during a vector read or stack push error the halt occurs on the corresponding fault handler for the vector error or stack push.
2. If a late arriving interrupt detected during a vector read or stack push error it is not taken. That is, an implementation that supports the late arrival optimization must suppress it in this case.

10.3 Core debug access example

If you want to halt the processor and write a value into one of the registers, perform the following sequence:

1. Write 0xA05F0003 to the Debug Halting Control and Status register. This enables debug and halts the core.
2. Wait for the **S_HALT** bit of the Debug Halting and Status Register to be set. This indicates that the core is halted.
3. Write the value that you want to be written to the Debug Core Register Data Register.
4. Write the register number that you want to write to into the Debug Core Register Selector Register.

10.4 Using application registers in core debug

You can also use the application registers for status access and to effect change on the system.

If you intend to use the application registers for core debug, be aware that:

- There are read-modify-write issues if both AHB-AP and the application are modifying these registers.
- For the write registers like PENDSET and PENDCLR, there are read-modify-write issues because these are not read first.
- For registers containing priority and other read-write registers, the register can change between the read and the write when performing a read-modify-write operation. In some cases the registers enable byte access to alleviate this situation, and the debugger must be aware of these issues when the processor is running.

Table 10-5 shows the application registers and the register bits that are most useful for use in core debug. For a complete list of the application registers see the *ARMv7-M Architecture Reference Manual*.

Table 10-5 Application registers for use in core debug

Register	Bits or fields for use in core debug
Interrupt Control State	ISRPREEMPT ISRPENDING VECTPENDING.
Vector Table Offset	To find vector table
Application Interrupt/Reset Control	VECTCLRACTIVE ENDIANESS
Configuration Control	DIV_0_TRP UNALIGN_TRP.
System Handler Control and State	ACTIVE PENDED

Chapter 11

System Debug

This chapter describes the processor system debug. It contains the following sections:

- *About system debug* on page 11-2
- *System debug access* on page 11-3
- *System debug programmer's model* on page 11-5
- *FPB* on page 11-6
- *DWT* on page 11-13
- *ITM* on page 11-30
- *AHB-AP* on page 11-39.

11.1 About system debug

The processor contains several system debug components that facilitate:

- low-cost debug
- trace and profiling
- breakpoints
- watchpoints
- code patching.

The system debug components are:

- *Flash Patch and Breakpoint* (FPB) unit to implement breakpoints and code patches.
- *Data Watchpoint and Trace* (DWT) unit to implement watchpoints, trigger resources, and system profiling.
- *Instrumentation Trace Macrocell* (ITM) for application-driven trace source that supports printf style debugging.
- *Embedded Trace Macrocell* (ETM) for instruction trace. The processor is supported in versions with and without the ETM.

All the debug components exist on the internal *Private Peripheral Bus* (PPB) and can be accessed using privileged code.

The system debug components are optional. You can remove all of the system debug components and have no trace or debug functionality. This means that breakpoints and watchpoints in addition to instrumentation trace and triggers might not be supported.

Note

- For a description of the Core debug, see Chapter 10 *Core Debug*.
-

11.2 System debug access

Debug control and data access occurs through the *Advanced High-performance Bus-Access Port* (AHB-AP) interface. This interface is driven by either the *Serial Wire Debug Port* (SW-DP) or *Serial Wire JTAG Debug Port* (SWJ-DP) components. See Chapter 13 *Debug Port* for information on the SW-DP and SWJ-DP components. Access includes:

- The internal PPB. Through this bus, the debugger can access components, including:
 - *Nested Vectored Interrupt Controller* (NVIC). Debug access to the processor core is made through the NVIC. For details, see Chapter 10 *Core Debug*.
 - DWT unit.
 - FPB unit.
 - ITM.
 - *Memory Protection Unit* (MPU).

Note

During a system reset the debugger can read all registers within the PPB space. It can also write to registers within the PPB space that are only reset by a power on reset.

- The External Private Peripheral Bus. Through this bus, debug can access:
 - ETM. A low-cost trace macrocell that supports instruction trace only. See Chapter 14 *Embedded Trace Macrocell* for more information.
 - *Trace Port Interface Unit* (TPIU). This component acts as a bridge between the Cortex-M3 trace data (from the ITM, and ETM if present) and an off-chip Trace Port Analyzer. See Chapter 17 *Trace Port Interface Unit* for more information.
 - ROM table.
- The DCode bus. Through this bus, debug can access memory located in code space.
- The System bus. Provides access to bus, memory, and peripherals located in system bus space.

Figure 11-1 on page 11-4 shows the structure of the system debug access, and shows how the AHB-AP can access each of the system components and external buses.

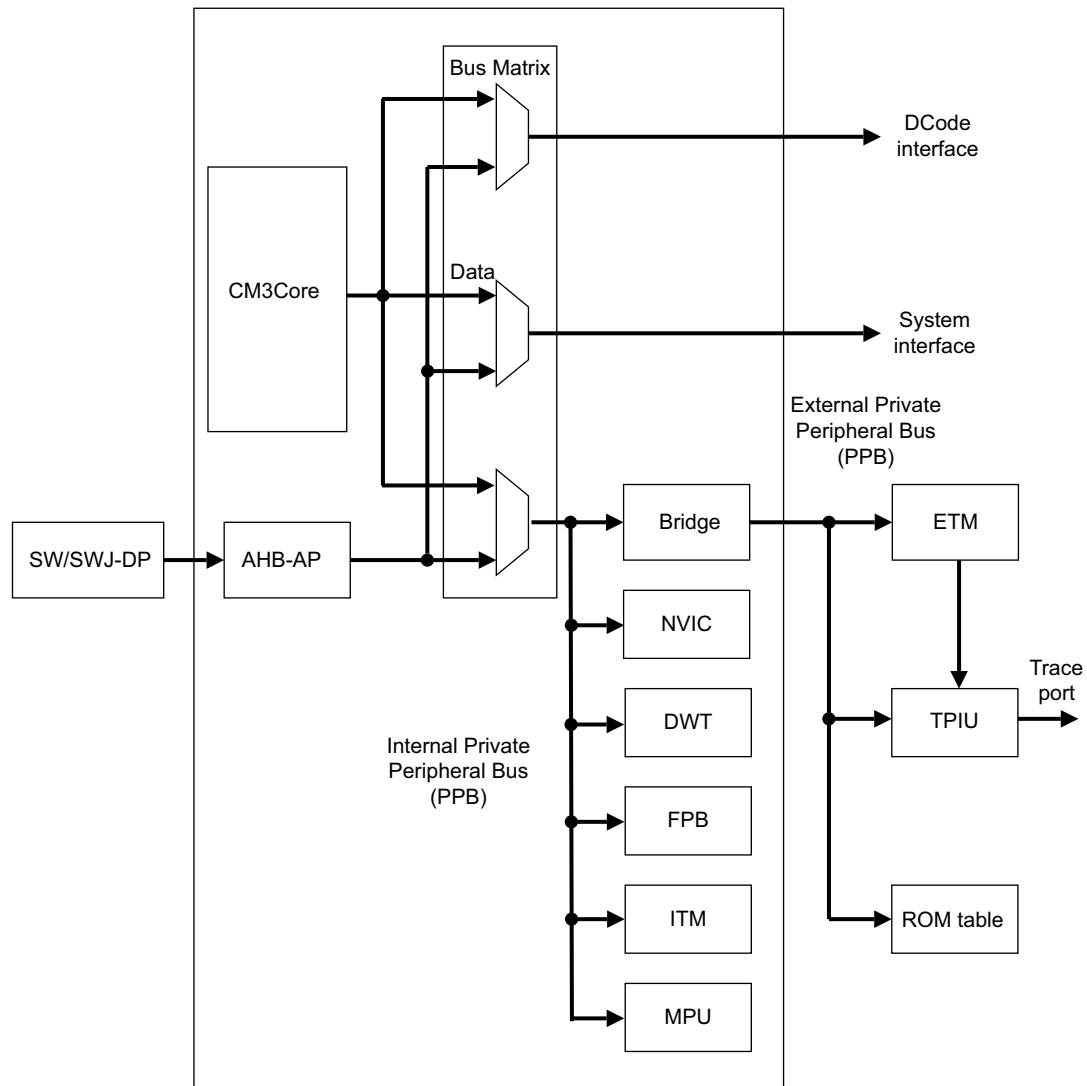


Figure 11-1 System debug access block diagram

11.3 System debug programmer's model

This section lists and describes the debug registers for all the system debug components. It contains:

- *FPB* on page 11-6
- *DWT* on page 11-13
- *ITM* on page 11-30
- *AHB-AP* on page 11-39.

Note

- For a description of the Core debug registers, see *Core debug registers* on page 10-3.
 - For a description of the SWJ-DP and SW-DP registers see Chapter 13 *Debug Port*.
 - For a description of the TPIU, see Chapter 17 *Trace Port Interface Unit*.
-

11.4 FPB

The FPB:

- implements hardware breakpoints
- patches code and data from code space to system space.

A full FPB unit contains:

- Two literal comparators for matching against literal loads from Code space, and remapping to a corresponding area in System space.
- Six instruction comparators for matching against instruction fetches from Code space, and remapping to a corresponding area in System space. Alternatively, you can individually configure the comparators to return a *Breakpoint Instruction* (BKPT) to the processor core on a match, so providing hardware breakpoint capability.

The FPB contains a global enable, but also individual enables for the eight comparators. If the comparison for an entry matches, the address is remapped to the address set in the remap register plus an offset corresponding to the comparator that matched, or is remapped to a BKPT instruction if that feature is enabled. The comparison happens dynamically, but the result of the comparison occurs too late to stop the original instruction fetch or literal load taking place from the Code space. The processor ignores this transaction however, and only the remapped transaction is used.

If an MPU is present, the MPU lookups are performed for the original address, not the remapped address.

You can remove the FPB if no debug is required or alternatively the number of breakpoints it supports can be reduced to two. If the FPB supports only two breakpoints then only comparators 0 and 1 are used, and flash patching is not supported.

———— Note ————

- Unaligned literal accesses are not remapped. The original access to the DCode bus takes place in this case.
- Load exclusives are Unpredictable to the FPB. The address is remapped but the access does not take place as an exclusive load.
- Remapping to the bit-band alias directly accesses the alias address, and does not remap to the bit-band region.

11.4.1 FPB programmer's model

Table 11-1 on page 11-7 lists the flash patch registers.

Note

You can configure any of the flash patch registers to be present or not present. Any register that is configured as not present reads as zero.

Table 11-1 FPB register summary

Name	Type	Address	Description
FP_CTRL	Read/write	0xE0002000	See <i>Flash Patch Control Register</i> on page 11-8
FP_REMAP	Read/write	0xE0002004	See <i>Flash Patch Remap Register</i> on page 11-9
FP_COMP0	Read/write	0xE0002008	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP1	Read/write	0xE000200C	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP2	Read/write	0xE0002010	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP3	Read/write	0xE0002014	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP4	Read/write	0xE0002018	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP5	Read/write	0xE000201C	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP6	Read/write	0xE0002020	See <i>Flash Patch Comparator Registers</i> on page 11-11
FP_COMP7	Read/write	0xE0002024	See <i>Flash Patch Comparator Registers</i> on page 11-11
PID4	Read-only	0xE0002FD0	Value 0x04
PID5	Read-only	0xE0002FD4	Value 0x00
PID6	Read-only	0xE0002FD8	Value 0x00
PID7	Read-only	0xE0002FDC	Value 0x00
PID0	Read-only	0xE0002FE0	Value 0x03
PID1	Read-only	0xE0002FE4	Value 0xB0
PID2	Read-only	0xE0002FE8	Value 0x2B
PID3	Read-only	0xE0002FEC	Value 0x00
CID0	Read-only	0xE0002FF0	Value 0x0D

Table 11-1 FPB register summary (continued)

Name	Type	Address	Description
CID1	Read-only	0xE0002FF4	Value 0xE0
CID2	Read-only	0xE0002FF8	Value 0x05
CID3	Read-only	0xE0002FFC	Value 0xB1

Flash Patch Control Register

Use the Flash Patch Control Register to enable the flash patch block.

The register address, access type, and Reset state are:

- Address 0xE0002000
- Access Read/write
- Reset state Bit [0] (ENABLE) is reset to 1'b0.

Figure 11-2 shows the bit assignments of the Flash Patch Control Register.

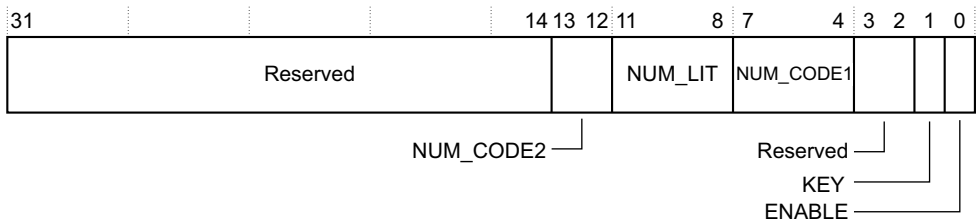


Figure 11-2 Flash Patch Control Register bit assignments

Table 11-2 describes the bit assignments of the Flash Patch Control Register.

Table 11-2 Flash Patch Control Register bit assignments

Bits	Field	Function
[31:15]	-	Reserved. Read As Zero. Write Ignored.
[14:12]	NUM_CODE2	Number of full banks of code comparators, sixteen comparators per bank. Where less than sixteen code comparators are provided, the bank count is zero, and the number present indicated by NUM_CODE. This read only field contains 3'b000 to indicate 0 banks for Cortex-M3 processor.
[11:8]	NUM_LIT	Number of literal slots field. This read only field contains either b0000 to indicate there are no literal slots or b0010 to indicate that there are two literal slots.

Table 11-2 Flash Patch Control Register bit assignments (continued)

Bits	Field	Function
[7:4]	NUM_CODE1	Number of code slots field. This read only field contains either b0000 to indicate that there are no code slots, b0010 to indicate that there are two code slots or b0110 to indicate that there are six code slots.
[3:2]	-	Reserved.
[1]	KEY	Key field. To write to the Flash Patch Control Register, you must write a 1 to this write-only bit.
[0]	ENABLE	Flash patch unit enable bit: 1 = flash patch unit enabled 0 = flash patch unit disabled. Reset clears the ENABLE bit.

Flash Patch Remap Register

Use the Flash Patch Remap Register to provide the location in System space where a matched address is remapped. The REMAP address is 8-word aligned, with one word allocated to each of the eight FPB comparators.

A comparison match remaps to:

{3'b001, REMAP, COMP[2:0], HADDR[1:0]}

where:

- 3'b001 hardwires the remapped access to system space
- REMAP is the 24-bit, 8-word aligned remap address

- COMP is the matching comparator. See Table 11-3.

Table 11-3 COMP mapping

COMP[2:0]	Comparator	Description
000	FP_COMP0	Instruction comparator
001	FP_COMP1	Instruction comparator
010	FP_COMP2	Instruction comparator
011	FP_COMP3	Instruction comparator
100	FP_COMP4	Instruction comparator
101	FP_COMP5	Instruction comparator
110	FP_COMP6	Literal comparator
111	FP_COMP7	Literal comparator

- **HADDR[1:0]** is the two *Least Significant Bits* (LSBs) of the original address.
HADDR[1:0] is always 2'b00 for instruction fetches.

The register address, access type, and Reset state are:

Address 0xE0002004
Access Read/write
Reset state This register is not reset

Figure 11-3 shows the bit assignments of the Flash Patch Remap Register.

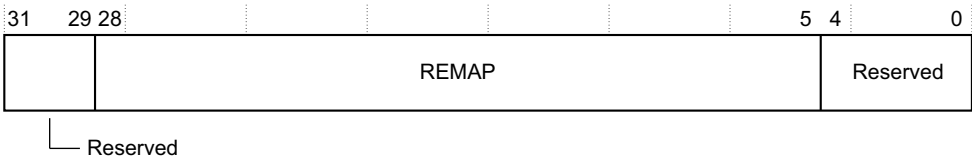


Figure 11-3 Flash Patch Remap Register bit assignments

Table 11-4 describes the bit assignments of the Flash Patch Remap Register.

Table 11-4 Flash Patch Remap Register bit assignments

Bits	Field	Function
[31:29]	-	Reserved. Read as b001. Hardwires the remap to the system space.
[28:5]	REMAP	Remap base address field.
[4:0]	-	Reserved. Read As Zero. Write Ignored.

Flash Patch Comparator Registers

Use the Flash Patch Comparator Registers to store the values to compare with the PC address.

The register address, access type, and Reset state are:

Access Read/write
Address 0xE0002008, 0xE000200C, 0xE0002010, 0xE0002014, 0xE0002018, 0xE000201C, 0xE0002020, 0xE0002024
Reset state Bit [0] (ENABLE) is reset to 1'b0.

Figure 11-4 shows the bit assignments of the Flash Patch Comparator Registers.

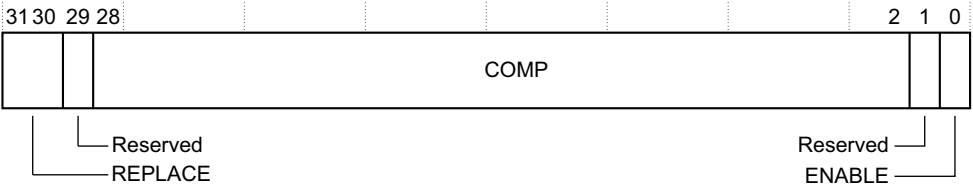


Figure 11-4 Flash Patch Comparator Registers bit assignments

Table 11-5 on page 11-12 describes the bit assignments of the Flash Patch Comparator Registers.

Table 11-5 Flash Patch Comparator Registers bit assignments

Bits	Field	Function
[31:30]	REPLACE	<p>This selects what happens when the COMP address is matched.</p> <p>It is interpreted as:</p> <p>b00 = remap to remap address. See FP_REMAP</p> <p>b01 = set BKPT on lower halfword, upper is unaffected</p> <p>b10 = set BKPT on upper halfword, lower is unaffected</p> <p>b11 = set BKPT on both lower and upper halfwords.</p> <p>Settings other than b00 are only valid for instruction comparators. Literal comparators ignore non-b00 settings.</p> <p>Address remapping only takes place for the b00 setting.</p>
[29]	-	Reserved
[28:2]	COMP	Comparison address.
[1]	-	Reserved.
[0]	ENABLE	<p>Compare and remap enable for Flash Patch Comparator Register <i>n</i>:</p> <p>1 = Flash Patch Comparator Register <i>n</i> compare and remap enabled</p> <p>0 = Flash Patch Comparator Register <i>n</i> compare and remap disabled.</p> <p>The ENABLE bit of FP_CTRL must also be set to enable comparisons.</p> <p>Reset clears the ENABLE bit.</p>

11.5 DWT

The DWT is an optional unit that performs the following debug functionality:

- It contains four comparators that you can configure as a hardware watchpoint, an ETM trigger, a PC sampler event trigger, or a data address sampler event trigger. The first comparator, DWT_COMP0, can also compare against the clock cycle counter, CYCCNT. The second comparator, DWT_COMP1, can also be used as a data comparator. You can configure the DWT to contain only one comparator that can be used as a watchpoint or as a trigger. If only one comparator is present then data matching is not supported.
 - The DWT contains counters for:
 - clock cycles (CYCCNT)
 - folded instructions
 - *Load Store Unit* (LSU) operations
 - sleep cycles
 - CPI (all instruction cycles except for the first cycle)
 - interrupt overhead.
- **Note** —————
- An event is emitted each time a counter overflows.
-
- You can configure the DWT to emit PC samples at defined intervals, and to emit interrupt event information.

11.5.1 Summary and description of the DWT registers

Table 11-6 on page 11-14 lists the DWT registers.

———— **Note** —————

You can configure any of the DWT registers to be present or not present. Any register that is configured as not present reads as zero.

Table 11-6 DWT register summary

Name	Type	Address	Reset value	Description
DWT_CTRL	Read/write	0xE0001000	0x40000000	See <i>DWT Control Register</i> on page 11-15
DWT_CYCCNT	Read/write	0xE0001004	0x00000000	See <i>DWT Current PC Sampler Cycle Count Register</i> on page 11-19
DWT_CPICNT	Read/write	0xE0001008	-	See <i>DWT CPI Count Register</i> on page 11-20
DWT_EXCCNT	Read/write	0xE000100C	-	See <i>DWT Exception Overhead Count Register</i> on page 11-20
DWT_SLEEPCNT	Read/write	0xE0001010	-	See <i>DWT Sleep Count Register</i> on page 11-21
DWT_LSUCNT	Read/write	0xE0001014	-	See <i>DWT LSU Count Register</i> on page 11-22
DWT_FOLDCNT	Read/write	0xE0001018	-	See <i>DWT Fold Count Register</i> on page 11-23
DWT_PCSR	Read-only	0xE000101C	-	See <i>DWT Program Counter Sample Register</i> on page 11-24
DWT_COMP0	Read/write	0xE0001020	-	See <i>DWT Comparator Registers</i> on page 11-24
DWT_MASK0	Read/write	0xE0001024	-	See <i>DWT Mask Registers 0-3</i> on page 11-25
DWT_FUNCTION0	Read/write	0xE0001028	0x00000000	See <i>DWT Function Registers 0-3</i> on page 11-25
DWT_COMP1	Read/write	0xE0001030	-	See <i>DWT Comparator Registers</i> on page 11-24
DWT_MASK1	Read/write	0xE0001034	-	See <i>DWT Mask Registers 0-3</i> on page 11-25
DWT_FUNCTION1	Read/write	0xE0001038	0x00000000	See <i>DWT Function Registers 0-3</i> on page 11-25
DWT_COMP2	Read/write	0xE0001040	-	See <i>DWT Comparator Registers</i> on page 11-24
DWT_MASK2	Read/write	0xE0001044	-	See <i>DWT Mask Registers 0-3</i> on page 11-25
DWT_FUNCTION2	Read/write	0xE0001048	0x00000000	See <i>DWT Function Registers 0-3</i> on page 11-25
DWT_COMP3	Read/write	0xE0001050	-	See <i>DWT Comparator Registers</i> on page 11-24
DWT_MASK3	Read/write	0xE0001054	-	See <i>DWT Mask Registers 0-3</i> on page 11-25
DWT_FUNCTION3	Read/write	0xE0001058	0x00000000	See <i>DWT Function Registers 0-3</i> on page 11-25
PID4	Read-only	0xE0001FD0	0x04	Value 0x04
PID5	Read-only	0xE0001FD4	0x00	Value 0x00

Table 11-6 DWT register summary (continued)

Name	Type	Address	Reset value	Description
PID6	Read-only	0xE0001FD8	0x00	Value 0x00
PID7	Read-only	0xE0001FDC	0x00	Value 0x00
PID0	Read-only	0xE0001FE0	0x02	Value 0x02
PID1	Read-only	0xE0001FE4	0xB0	Value 0xB0
PID2	Read-only	0xE0001FE8	0x2B	Value 0x2B
PID3	Read-only	0xE0001FEC	0x00	Value 0x00
CID0	Read-only	0xE0001FF0	0x0D	Value 0x0D
CID1	Read-only	0xE0001FF4	0xE0	Value 0xE0
CID2	Read-only	0xE0001FF8	0x05	Value 0x05
CID3	Read-only	0xE0001FFC	0xB1	Value 0xB1

DWT Control Register

Use the DWT Control Register to enable the DWT unit.

The register address, access type, and Reset state are:

Address 0xE0001000

Access Read/write

Reset state 0x40000000 if four comparators for watchpoints and triggers are present
 0x4F000000 if four comparators for watchpoints only are present
 0x10000000 if only one comparator is present
 0x1F000000 if one comparator for watchpoints and not triggers is present
 0x00000000 if DWT is not present.

Figure 11-5 on page 11-16 shows the bit assignments of the DWT Control Register.

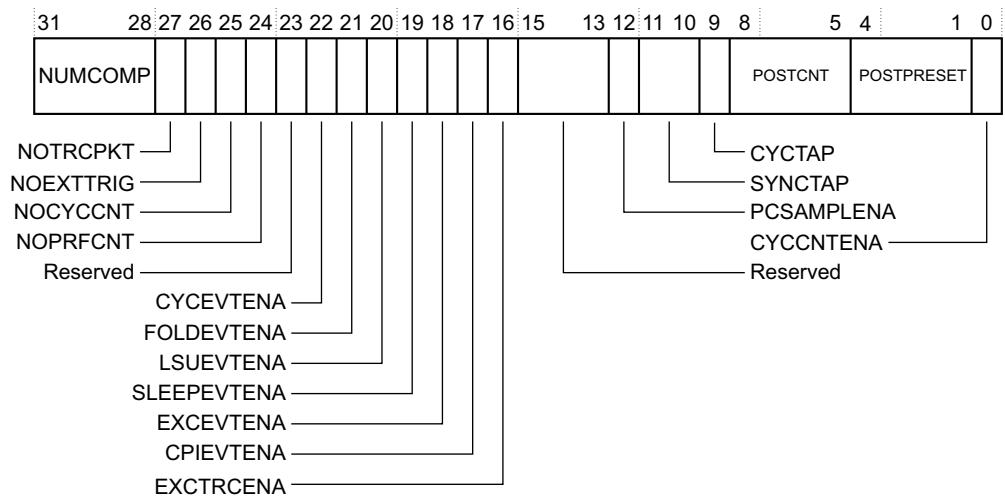


Figure 11-5 DWT Control Register bit assignments

Table 11-7 describes the bit assignments of the DWT Control Register.

Table 11-7 DWT Control Register bit assignments

Bits	Field	Function
[31:28]	NUMCOMP	Number of comparators field. This read-only field contains the number of comparators present. Valid values are b0100, b0001, or b0000.
[27]	NOTRCPKT	When set, trace sampling and exception tracing are not supported.
[26]	NOEXTTRIG	When set, no CMPMATCH[N] support.
[25]	NOCYCCNT	When set, DWT_CYCCNT is not supported.
[24]	NOPRFCNT	When set, DWT_FOLDCNT, DWT_LSUCNT, DWT_SLEEPcnt, DWT_EXCCNT, and DWT_CPICNT are not supported.
[23]	Reserved	-
[22]	CYCEVTENA	Enables Cycle count event. Emits an event when the POSTCNT counter triggers it. See CYCTAP (bit [9]) and POSTPRESET, bits [4:1], for details. 1 = Cycle count events enabled 0 = Cycle count events disabled. This event is only emitted if PCSAMPLENA, bit [12], is disabled. PCSAMPLENA overrides the setting of this bit. Reset clears the CYCEVTENA bit.

Table 11-7 DWT Control Register bit assignments (continued)

Bits	Field	Function
[21]	FOLDEVTENA	<p>Enables Folded instruction count event. Emits an event when DWT_FOLDCNT overflows (every 256 cycles of folded instructions). A folded instruction is one that does not incur even one cycle to execute. For example, an IT instruction is folded away and so does not use up one cycle.</p> <p>1 = Folded instruction count events enabled. 0 = Folded instruction count events disabled.</p> <p>Reset clears the FOLDEVTENA bit.</p>
[20]	LSUEVTENA	<p>Enables LSU count event. Emits an event when DWT_LSUCNT overflows (every 256 cycles of LSU operation). LSU counts include all LSU costs after the initial cycle for the instruction.</p> <p>1 = LSU count events enabled. 0 = LSU count events disabled.</p> <p>Reset clears the LSUEVTENA bit.</p>
[19]	SLEEPEVTENA	<p>Enables Sleep count event. Emits an event when DWT_SLEPCNT overflows (every 256 cycles that the processor is sleeping).</p> <p>1 = Sleep count events enabled. 0 = Sleep count events disabled.</p> <p>Reset clears the SLEEPEVTENA bit.</p>
[18]	EXCEVTENA	<p>Enables Interrupt overhead event. Emits an event when DWT_EXCCNT overflows (every 256 cycles of interrupt overhead).</p> <p>1 = Interrupt overhead event enabled. 0 = Interrupt overhead event disabled.</p> <p>Reset clears the EXCEVTENA bit.</p>
[17]	CPIEVTENA	<p>Enables CPI count event. Emits an event when DWT_CPICNT overflows (every 256 cycles of multi-cycle instructions).</p> <p>1 = CPI counter events enabled. 0 = CPI counter events disabled.</p> <p>Reset clears the CPIEVTENA bit.</p>
[16]	EXCTRCENA	<p>Enables Interrupt event tracing:</p> <p>1 = interrupt event trace enabled 0 = interrupt event trace disabled.</p> <p>Reset clears the EXCEVTENA bit.</p>
[15:13]	-	Reserved

Table 11-7 DWT Control Register bit assignments (continued)

Bits	Field	Function
[12]	PCSAMPLEENA	<p>Enables PC Sampling event. A PC sample event is emitted when the POSTCNT counter triggers it. See CYCTAP, bit [9], and POSTPRESET, bits [4:1], for details. Enabling this bit overrides CYCEVTENA (bit [20]).</p> <p>1 = PC Sampling event enabled.</p> <p>0 = PC Sampling event disabled.</p> <p>Reset clears the PCSAMPLEENA bit.</p>
[11:10]	SYNCTAP	<p>Feeds a synchronization pulse to the ITM SYNCENA control. The value selected here picks the rate (approximately 1/second or less) by selecting a tap on the DWT_CYCCNT register. To use synchronization (heartbeat and hot-connect synchronization), CYCCNTENA must be set to 1, SYNCTAP must be set to one of its values, and SYNCENA must be set to 1.</p> <p>0b00 = Disabled. No synch counting.</p> <p>0b01 = Tap at CYCCNT bit 24.</p> <p>0b10 = Tap at CYCCNT bit 26.</p> <p>0b11 = Tap at CYCCNT bit 28.</p>
[9]	CYCTAP	<p>Selects a tap on the DWT_CYCCNT register. These are spaced at bits [6] and [10]:</p> <p>CYCTAP = 0 selects bit [6] to tap</p> <p>CYCTAP = 1 selects bit [10] to tap.</p> <p>When the selected bit in the CYCCNT register changes from 0 to 1 or 1 to 0, it emits into the POSTCNT, bits [8:5], post-scalar counter. That counter then counts down. On a bit change when post-scalar is 0, it triggers an event for PC sampling or CYCEVTCNT.</p>
[8:5]	POSTCNT	<p>Post-scalar counter for CYCTAP.</p> <p>When the selected tapped bit changes from 0 to 1 or 1 to 0, the post scalar counter is down-counted when not 0.</p> <p>If 0, it triggers an event for PCSAMPLEENA or CYCEVTENA use. It also reloads with the value from POSTPRESET (bits [4:1]).</p>
[4:1]	POSTPRESET	<p>Reload value for POSTCNT, bits [8:5], post-scalar counter.</p> <p>If this value is 0, events are triggered on each tap change (a power of 2, such as $1 \ll 6$ or $1 \ll 10$).</p> <p>If this field has a non-0 value, this forms a count-down value, to be reloaded into POSTCNT each time it reaches 0. For example, a value 1 in this register means an event is formed every other tap change.</p>
[0]	CYCCNTENA	<p>Enable the CYCCNT counter. If not enabled, the counter does not count and no event is generated for PS sampling or CYCCNTENA. In normal use, the debugger must initialize the CYCCNT counter to 0.</p>

Note

The TRCENA bit of the Debug Exception and Monitor Control Register must be set before you can use the DWT. See *Debug Exception and Monitor Control Register* on page 10-8.

Note

The DWT is enabled independently from the ITM. If you enable the DWT to emit events, you must also enable the ITM.

DWT Current PC Sampler Cycle Count Register

Use the DWT Current PC Sampler Cycle Count Register to count the number of core cycles. This count can measure elapsed execution time.

The register address, access type, and Reset state are:

Address 0xE0001004

Access Read/write

Reset state 0x00000000

Table 11-8 describes the bit assignments of the DWT Current PC Sampler Cycle Count Register.

Table 11-8 DWT Current PC Sampler Cycle Count Register bit assignments

Bits	Field	Function
[31:0]	CYCCNT	Current PC Sampler Cycle Counter count value. When enabled, this counter counts the number of core cycles, except when the core is halted. CYCCNT is a free running counter, counting upwards. It wraps around to 0 on overflow. The debugger must initialize this to 0 when first enabling.

This is a free-running counter. The counter has three functions:

- When PCSAMPLENA is set, the PC is sampled and emitted when the selected tapped bit changes value (0 to 1 or 1 to 0) and any post-scalar value counts to 0.
- When CYCEVTENA is set (and PCSAMPLENA is clear), an event is emitted when the selected tapped bit changes value (0 to 1 or 1 to 0) and any post-scalar value counts to 0.

- Applications and debuggers can use the counter to measure elapsed execution time. By subtracting a start and an end time, an application can measure time between in-core clocks (other than when Halted in debug). This is valid to 2^{32} core clock cycles (for example, almost 86 seconds at 50MHz).

DWT CPI Count Register

Use the DWT CPI Count Register to count the total number of instruction cycles beyond the first cycle.

The register address, access type, and Reset state are:

Address 0xE0001008
Access Read-write
Reset state -

Figure 11-6 shows the bit assignments of the DWT CPI Count Register.

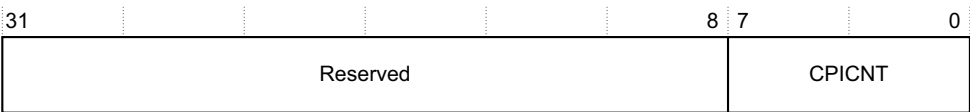


Figure 11-6 DWT CPI Count Register bit assignments

Table 11-9 describes the bit assignments of the DWT CPI Count Register.

Table 11-9 DWT CPI Count Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	CPICNT	Current CPI counter value. Increments on the additional cycles (the first cycle is not counted) required to execute all instructions except those recorded by DWT_LSUCNT. This counter also increments on all instruction fetch stalls. If CPIEVTENA is set, an event is emitted when the counter overflows. Clears to 0 on enabling.

DWT Exception Overhead Count Register

Use the DWT Exception Overhead Count Register to count the total cycles spent in interrupt processing.

The register address, access type, and Reset state are:

Address 0xE000100C

Copyright © 2005-2008 ARM Limited. All rights reserved.
Non-Confidential

11-21



Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	EXCCNT	Current interrupt overhead counter value. Counts the total cycles spent in interrupt processing (for example entry stacking, return unstacking, pre-emption). An event is emitted on counter overflow (every 256 cycles). This counter initializes to 0 when enabled. Clears to 0 on enabling.

Use the DWT Sleep Count Register to count the total number of cycles during which the processor is sleeping.

Address	0xE0001010
Access	Read-write
Reset state	-

31		8	7	0
Reserved			SLEEPNT	

Table 11-11 describes the bit assignments of the DWT Sleep Count Register.

Table 11-11 DWT Sleep Count Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	SLEEP_CNT	Sleep counter. Counts the number of cycles during which the processor is sleeping. An event is emitted on counter overflow (every 256 cycles). This counter initializes to 0 when enabled.

————— **Note** —————

SLEEP_CNT is clocked using **FCLK**. It is possible that the frequency of **FCLK** might be reduced while the processor is sleeping to minimize power consumption. This means that sleep duration must be calculated with the frequency of **FCLK** during sleep.

DWT LSU Count Register

Use the DWT LSU Count Register to count the total number of cycles during which the processor is processing an LSU operation beyond the first cycle.

The register address, access type, and Reset state are:

Address 0xE0001014
Access Read/write
Reset state -

Figure 11-9 describes the bit assignments of the DWT LSU Count Register.



Figure 11-9 DWT LSU Count Register bit assignments

Table 11-12 describes the bit assignments of the DWT LSU Count Register.

Table 11-12 DWT LSU Count Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	LSUCNT	<p>LSU counter. This counts the total number of cycles that the processor is processing an LSU operation. The initial execution cost of the instruction is not counted.</p> <p>For example, an LDR that takes two cycles to complete increments this counter one cycle. Equivalently, an LDR that stalls for two cycles (and so takes four cycles), increments this counter three times. An event is emitted on counter overflow (every 256 cycles).</p> <p>Clears to 0 on enabling.</p>

DWT Fold Count Register

Use the DWT Fold Count Register to count the total number of folded instructions. This counts 1 for each instruction that takes 0 cycles.

The register address, access type, and Reset state are:

Address 0xE0001018

Access Read/write

Reset state -

Figure 11-10 describes the bit assignments of the DWT Fold Count Register.

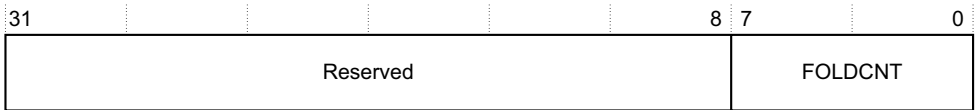


Figure 11-10 DWT Fold Count Register bit assignments

Table 11-13 describes the bit assignments of the DWT Fold Count Register.

Table 11-13 DWT Fold Count Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	FOLDCNT	This counts the total number folded instructions. This counter initializes to 0 when enabled.

DWT Program Counter Sample Register

Use the DWT *Program Counter Sample Register* (PCSR) to enable coarse-grained software profiling using a debug agent, without changing the currently executing code.

If the core is not in debug state, the value returned is the instruction address of a recently executed instruction.

If the core is in debug state, the value returned is 0xFFFFFFFF.

The register address, access type, and Reset state are:

Address 0xE000101C
Access Read-only
Reset state Unpredictable

DWT Program Counter Sample Register bit assignments describes the field of the DWT PCSR.

Table 11-14 DWT Program Counter Sample Register bit assignments

Bits	Field	Function
[31:0]	EIASAMPLE	Execution instruction address sample, or 0xFFFFFFFF if the core is halted.

DWT Comparator Registers

Use the DWT Comparator Registers 0-3 to write the values that trigger watchpoint events.

The register address, access type, and Reset state are:

Address 0xE0001020, 0xE0001030, 0xE0001040, 0xE0001050
Access Read/write
Reset state -

Table 11-15 describes the field of DWT Comparator Registers 0-3.

Table 11-15 DWT Comparator Registers 0-3 bit assignments

Field	Name	Definition
[31:0]	COMP	Data value to compare against PC and the data address as given by DWT_FUNCTIONx. DWT_COMP0 can also compare against the value of the PC Sampler Counter (DWT_CYCCNT). DWT_COMP1 can also compare against data values so that data matching can be performed (DATAVMATCH).

DWT Mask Registers 0-3

Use the DWT Mask Registers 0-3 to apply a mask to data addresses when matching against COMP.

The register address, access type, and Reset state are:

Address 0xE0001024, 0xE0001034, 0xE0001044, 0xE0001054
Access Read/write
Reset state -

Figure 11-11 shows the bit assignments of DWT Mask Registers 0-3.

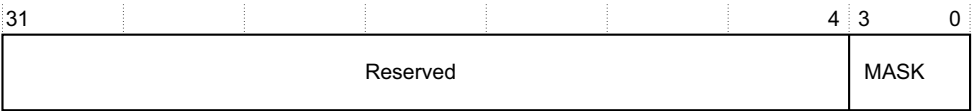


Figure 11-11 DWT Mask Registers 0-3 bit assignments

Table 11-16 describes the bit assignments of DWT Mask Registers 0-3.

Table 11-16 DWT Mask Registers 0-3 bit assignments

Bits	Field	Function
[31:4]	-	Reserved.
[3:0]	MASK	Mask on data address when matching against COMP. This is the size of the ignore mask. So, $\sim 0 < \text{MASK}$ forms the mask against the address to use. That is, DWT matching is performed as: $(\text{ADDR} \& (\sim 0 < \text{MASK})) == \text{COMP}$ However, the actual comparison is slightly more complex to enable matching an address wherever it appears on a bus. So, if COMP is 3, this matches a word access of 0, because 3 would be within the word.

DWT Function Registers 0-3

Use the DWT Function Registers 0-3 to control the operation of the comparator. Each comparator can:

- Match against either the PC or the data address. This is controlled by CYCMATCH. This function is only available for comparator 0 (DWT_COMP0).
- Perform data value comparisons if associated address comparators have performed an address match. This function is only available for comparator 1 (DWT_COMP1).

- Emit data or PC couples, trigger the ETM, or generate a watchpoint depending on the operation defined by FUNCTION.

The register address, access type, and Reset state are:

Address 0xE0001028, 0xE0001038, 0xE0001048, 0xE0001058
Access Read/write
Address 0x00000000

Figure 11-12 shows the bit assignments of DWT Function Registers 0-3.

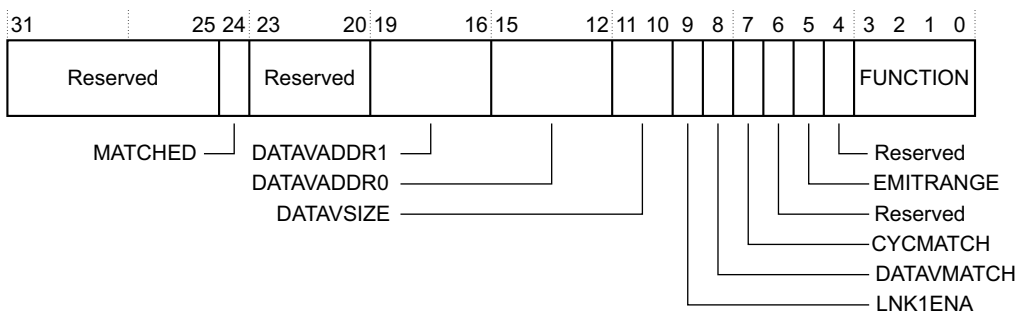


Figure 11-12 DWT Function Registers 0-3 bit assignments

Table 11-17 describes the bit assignments of DWT Function Registers 0-3.

Table 11-17 Bit functions of DWT Function Registers 0-3

Bits	Field	Function
[31:25]	-	Reserved.
[24]	MATCHED	This bit is set when the comparator matches, and indicates that the operation defined by FUNCTION has occurred since this bit was last read. This bit is cleared on read.
[23:20]	-	Reserved.
[19:16]	DATAVADDR1	Identity of a second linked address comparator for data value matching when DATAVMATCH == 1 and LNK1ENA == 1.
[15:12]	DATAVADDR0	Identity of a linked address comparator for data value matching when DATAVMATCH == 1.

Table 11-17 Bit functions of DWT Function Registers 0-3 (continued)

Bits	Field	Function
[11:10]	DATAVSIZE	Defines the size of the data in the COMP register that is to be matched: 00 = byte 01 = halfword 10 = word 11 = Unpredictable.
[9]	LNK1ENA	Read-only: 0 = DATAVADDR1 not supported 1 = DATAVADDR1 supported (enabled).
[8]	DATAVMATCH	This bit is only available in comparator 1. When DATAVMATCH is set, this comparator performs data value compares. The comparators given by DATAVADDR0 and DATAVADDR1 provide the address for the data comparison. If DATAVMATCH is set in DWT_FUNCTION1, the FUNCTION setting for the comparators given by DATAVADDR0 and DATAVADDR1 are overridden and those comparators only provide the address match for the data comparison.
[7]	CYCMATCH	Only available in comparator 0. When set, this comparator compares against the clock cycle counter.
[6]	-	Reserved.
[5]	EMITRANGE	Emit range field. Reserved to permit emitting offset when range match occurs. Reset clears the EMITRANGE bit. PC sampling is not supported when EMITRANGE is enabled. EMITRANGE only applies for: FUNCTION = b0001, b0010, b0011, b1100, b1101, b1110, and b1111.
[4]	-	Reserved.
[3:0]	FUNCTION	See Table 11-18 on page 11-28 for FUNCTION settings.

Table 11-18 describes the function settings of the DWT Function Registers.

Table 11-18 Settings for DWT Function Registers

Value	Function
b0000	Disabled
b0001	EMITRANGE = 0, sample and emit PC through ITM EMITRANGE = 1, emit address offset through ITM
b0010	EMITRANGE = 0, emit data through ITM on read and write. EMITRANGE = 1, emit data and address offset through ITM on read or write.
b0011	EMITRANGE = 0, sample PC and data value through ITM on read or write. EMITRANGE = 1, emit address offset and data value through ITM on read or write.
b0100	Watchpoint on PC match.
b0101	Watchpoint on read.
b0110	Watchpoint on write.
b0111	Watchpoint on read or write.
b1000	ETM trigger on PC match
b1001	ETM trigger on read
b1010	ETM trigger on write
b1011	ETM trigger on read or write
b1100	EMITRANGE = 0, sample data for read transfers EMITRANGE = 1, sample Daddr [15:0] for read transfers
b1101	EMITRANGE = 0, sample data for write transfers EMITRANGE = 1, sample Daddr [15:0] for write transfers
b1110	EMITRANGE = 0, sample PC + data for read transfers EMITRANGE = 1, sample Daddr [15:0] + data for read transfers
b1111	EMITRANGE = 0, sample PC + data for write transfers EMITRANGE = 1, sample Daddr [15:0] + data for write transfers

———— **Note** ————

- If the ETM is not fitted, then ETM trigger is not possible.

- Data value is only sampled for accesses that do not fault (MPU or bus fault). The PC is sampled irrespective of any faults. The PC is only sampled for the first address of a burst.
 - FUNCTION is overridden for comparators given by DATAVADDR0 and DATAVADDR1 in DWT_FUNCTION1 if DATAVMATCH is also set in DWT_FUNCTION1. The comparators given by DATAVADDR0 and DATAVADDR1 can then only perform address comparator matches for comparator 1 data matches.
 - If the data matching functionality is not included during implementation it is not possible to set DATAVADDR0, DATAVADDR1, or DATAVMATCH in DWT_FUNCTION1. This means that the data matching functionality is not available in the implementation. Test the availability of data matching by writing and reading the DATAVMATCH bit in DWT_FUNCTION1. If it is not settable then data matching is unavailable.
 - PC match is not recommended for watchpoints because it stops after the instruction. It mainly guards and triggers the ETM.
-

11.6 ITM

The ITM is a an optional application driven trace source that supports printf style debugging to trace *Operating System* (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets. There are three sources that can generate packets. If multiple sources generate packets at the same time, the ITM arbitrates the order in which packets are output. The three sources in decreasing order of priority are:

- Software trace. Software can write directly to ITM stimulus registers. This emits packets.
- Hardware trace. The DWT generates these packets, and the ITM emits them.
- Time stamping. Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp. The Cortex-M3 clock or the bitclock rate of the *Serial Wire Viewer* (SWV) output clocks the counter.

11.6.1 Summary and description of the ITM registers

———— Note —————

TRCENA of the Debug Exception and Monitor Control Register must be enabled before you program or use the ITM, see *Debug Exception and Monitor Control Register* on page 10-8.

Table 11-19 lists the ITM registers.

———— Note —————

You can configure any of the ITM registers to be present or not present. Any register that is configured as not present reads as zero.

Table 11-19 ITM register summary

Name	Type	Address	Reset value	Description
Stimulus Ports 0-31	Read/write	0xE0000000-0xE000007C	-	See <i>ITM Stimulus Ports 0-31</i> on page 11-32
Trace Enable	Read/write	0xE0000E00	0x00000000	See <i>ITM Trace Enable Register</i> on page 11-32
Trace Privilege	Read/write	0xE0000E40	0x00000000	See <i>ITM Trace Privilege Register</i> on page 11-33

Table 11-19 ITM register summary (continued)

Name	Type	Address	Reset value	Description
Trace Control Register	Read/write	0xE0000E80	0x00000000	See <i>ITM Trace Control Register</i> on page 11-34
Integration Write	Write-only	0xE0000EF8	0x00000000	See <i>ITM Integration Write Register</i> on page 11-35
Integration Read	Read-only	0xE0000EFC	0x00000000	See <i>ITM Integration Read Register</i> on page 11-36
Integration Mode Control	Read/write	0xE0000F00	0x00000000	See <i>ITM Integration Mode Control Register</i> on page 11-36
Lock Access Register	Write-only	0xE0000FB0	0x00000000	See <i>ITM Lock Access Register</i> on page 11-37
Lock Status Register	Read-only	0xE0000FB4	0x00000003	See <i>ITM Lock Status Register</i> on page 11-37
PID4	Read-only	0xE0000FD0	0x00000004	Value 0x04
PID5	Read-only	0xE0000FD4	0x00000000	Value 0x00
PID6	Read-only	0xE0000FD8	0x00000000	Value 0x00
PID7	Read-only	0xE0000FDC	0x00000000	Value 0x00
PID0	Read-only	0xE0000FE0	0x00000001	Value 0x01
PID1	Read-only	0xE0000FE4	0x000000B0	Value 0xB0
PID2	Read-only	0xE0000FE8	0x0000002B	Value 0x2B
PID3	Read-only	0xE0000FEC	0x00000000	Value 0x00
CID0	Read-only	0xE0000FF0	0x0000000D	Value 0x0D
CID1	Read-only	0xE0000FF4	0x000000E0	Value 0xE0
CID2	Read-only	0xE0000FF8	0x00000005	Value 0x05
CID3	Read-only	0xE0000FFC	0x000000B1	Value 0xB1

———— **Note** ————

ITM registers are fully accessible in privileged mode. In user mode, all registers can be read, but only the Stimulus Registers and Trace Enable Registers can be written, and only when the corresponding Trace Privilege Register bit is set. Invalid user mode writes to the ITM registers are discarded.

ITM Stimulus Ports 0-31

Each of the 32 stimulus ports has its own address. A write to one of these locations causes data to be written into the FIFO if the corresponding bit in the Trace Enable Register is set. Reading from any of the stimulus ports returns the FIFO status in bit [0]:

- 0 = full
- 1 = not full.

The polled FIFO interface does not provide an atomic read-modify-write, so you must use the Cortex-M3 exclusive monitor if a polled printf is used concurrently with ITM usage by interrupts or other threads. The following polled code guarantees stimulus is not lost by polled access to the ITM:

```
; r0 = Value to write to port
; r1 and r2 = Temporary scratch registers

      MOV r1, #0xE0000000      ; r1 = Stimulus port base
Retry LDREX r2, [r1, #Port*4] ; Load FIFO full status
      CMP r2, #0                ; Compare with full
      ITT NE                     ; If (not full)
      STREXNE r2, [r1, #Port*4]; Try sending value to port
      CMPNE r2, #1              ; and check for failure
      BEQ Retry                 ; If full or failed then retry
```

ITM Trace Enable Register

Use the Trace Enable Register to generate trace data by writing to the corresponding stimulus port.

The register address, access type, and Reset state are:

Access	Read/write
Address	0xE0000E00
Reset	0x00000000

Table 11-20 describes the field of the ITM Trace Enable Register.

Table 11-20 ITM Trace Enable Register bit assignments

Bits	Field	Function
[31:0]	STIMENA	Bit mask to enable tracing on ITM stimulus ports. One bit per stimulus port.

Note

Privileged writes are accepted to this register if ITMENA is set. User writes are accepted to this register if ITMENA is set and the appropriate privilege mask is cleared. Privileged access to the stimulus ports enables an RTOS kernel to guarantee instrumentation slots or bandwidth as required.

ITM Trace Privilege Register

Use the ITM Trace Privilege Register to enable an operating system to control which stimulus ports are accessible by user code.

Note

You can only write to this register in privileged mode.

The register address, access type, and Reset state are:

Access	Read/write
Address	0xE0000E40
Reset	0x00000000

Figure 11-13 shows the ITM Trace Privilege Register bit assignments.

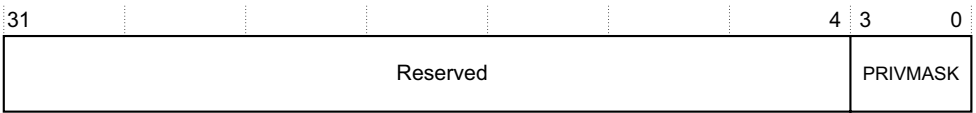


Figure 11-13 ITM Trace Privilege Register bit assignments

Table 11-21 describes the bit assignments of the ITM Trace Privilege Register.

Table 11-21 ITM Trace Privilege Register bit assignments

Bits	Field	Function
[31:4]	-	Reserved.
[3:0]	PRIVMASK	Bit mask to enable tracing on ITM stimulus ports: bit [0] = stimulus ports [7:0] bit [1] = stimulus ports [15:8] bit [2] = stimulus ports [23:16] bit [3] = stimulus ports [31:24].

ITM Trace Control Register

Use this register to configure and control ITM transfers.

———— Note —————

You can only write to this register in privilege mode.

The register address, access type, and Reset state are:

Access Read/write
Address 0xE0000E80
Reset 0x00000000

Figure 11-14 shows the ITM Control Register bit assignments.

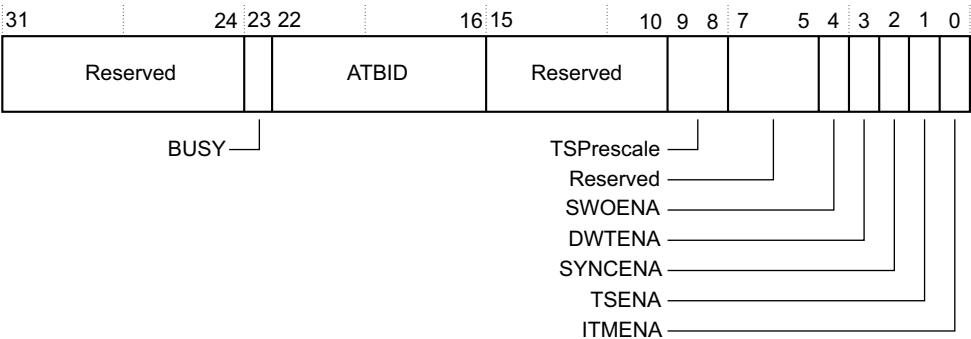


Figure 11-14 ITM Trace Control Register bit assignments

Table 11-22 describes the bit assignments of the ITM Control Register.

Table 11-22 ITM Trace Control Register bit assignments

Bits	Field	Function
[31:24]	-	0b00000000.
[23]	BUSY	Set when ITM events present and being drained
[22:16]	ATBID	ATB ID for CoreSight system.
[15:10]	-	0b000000.

Table 11-22 ITM Trace Control Register bit assignments (continued)

Bits	Field	Function
[9:8]	TSPrescale	Timestamp prescaler: 0b00 = no prescaling 0b01 = divide by 4 0b10 = divide by 16 0b11 = divide by 64.
[7:5]	-	Reserved.
[4]	SWOENA	Enable SWV behavior – count on TPIUACTV and TPIUBAUD.
[3]	DWTENA	Enables the DWT stimulus.
[2]	SYNCENA	Enables sync packets for TPIU.
[1]	TSENA	Enables differential timestamps. Differential timestamps are emitted when a packet is written to the FIFO with a non-zero timestamp counter, and when the timestamp counter overflows. Timestamps are emitted during idle times after a fixed number of two million cycles. This provides a time reference for packets and inter-packet gaps. If SWOENA (bit [4]) is set, timestamps are triggered by activity on the internal trace bus only. In this case there is no regular timestamp output when the ITM is idle.
[0]	ITMENA	Enable ITM. This is the master enable, and must be set before ITM Stimulus and Trace Enable registers can be written.

Note

DWT is not enabled in the ITM block. However, DWT stimulus entry into the FIFO is controlled by DWTENA. If DWT requires timestamping, the TSSSEN bit must be set.

ITM Integration Write Register

Use this register to determine the behavior of the ATVALIDM bit.

Figure 11-15 shows the ITM Integration Write Register bit assignments.

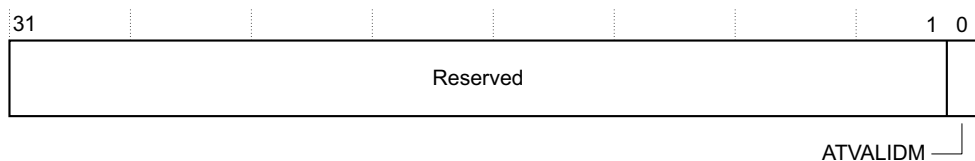


Figure 11-15 ITM Integration Write Register bit assignments

Table 11-23 describes the bit assignments of the ITM Integration Write Register.

Table 11-23 ITM Integration Write Register bit assignments

Bits	Field	Function
[31:1]	-	Reserved
[0]	ATVALIDM	When the integration mode is set: 0 = ATVALIDM clear 1 = ATVALIDM set.

Note
Bit [0] drives ATVALIDM when mode is set.

ITM Integration Read Register

Use this register to read the value on ATREADYM

Figure 11-16 shows the ITM Integration Read Register bit assignments.

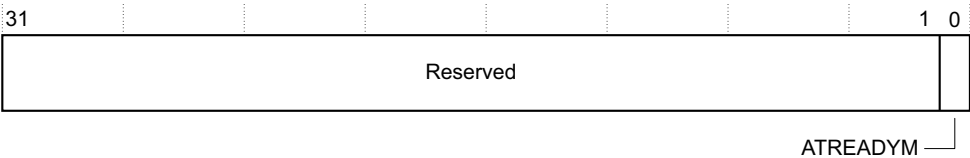


Figure 11-16 ITM Integration Read Register bit assignments

Table 11-24 describes the bit assignments of the ITM Integration Read Register.

Table 11-24 ITM Integration Read Register bit assignments

Bits	Field	Function
[31:1]	-	Reserved
[0]	ATREADYM	Value on ATREADYM

ITM Integration Mode Control Register

Use this register to enable write accesses to the Control Register.

Figure 11-17 on page 11-37 shows the ITM Integration Mode Control Register bit assignments.

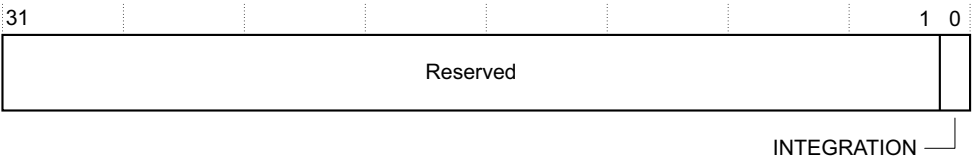


Figure 11-17 ITM Integration Mode Control bit assignments

Table 11-25 describes the bit assignments of the ITM Integration Mode Control Register

Table 11-25 ITM Integration Mode Control Register bit assignments

Bits	Field	Function
[31:1]	-	Reserved
[0]	INTEGRATION	0 = ATVALIDM normal 1 = ATVALIDM driven from Integration Write Register

ITM Lock Access Register

Use this register to prevent write accesses to the Control Register.

Table 11-26 describes the bit assignments of the ITM Lock Access Register

Table 11-26 ITM Lock Access Register bit assignments

Bits	Field	Function
[31:0]	Lock Access	A privileged write of 0xC5ACCE55 enables more write access to Control Register 0xE00::0xFFC. An invalid write removes write access.

ITM Lock Status Register

Use this register to enable write accesses to the Control Register.

Figure 11-18 on page 11-38 shows the ITM Lock Status Register bit assignments.

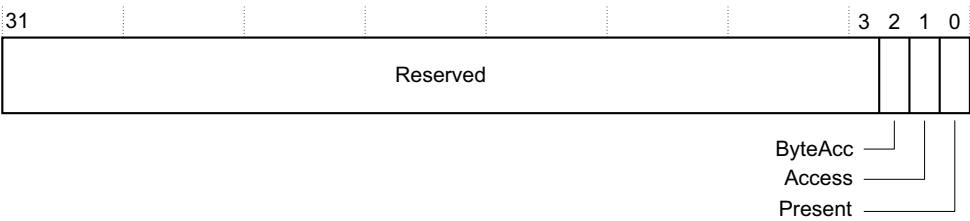


Figure 11-18 ITM Lock Status Register bit assignments

Table 11-27 describes the bit assignments of the ITM Lock Status Register

Table 11-27 ITM Lock Status Register bit assignments

Bits	Field	Function
[31:3]	-	Reserved.
[2]	ByteAcc	You cannot implement 8-bit lock accesses.
[1]	Access	Write access to component is blocked. All writes are ignored, reads are permitted.
[0]	Present	Indicates that a lock mechanism exists for this component.

11.7 AHB-AP

AHB-AP is an optional debug access port into the Cortex-M3 system, and provides access to all memory and registers in the system, including processor registers through the NVIC. System access is independent of the processor status. Either SW-DP or SWJ-DP accesses AHB-AP.

AHB-AP is a master into the Bus Matrix. Transactions are made using the AHB-AP programmer's model, which generates AHB-Lite transactions into the Bus Matrix. See *Summary and description of the AHB-AP registers*.

11.7.1 AHB-AP transaction types

AHB-AP does not do back-to-back transactions on the bus, and so all transactions are non-sequential. AHB-AP can perform unaligned and bit-band transactions. The Bus Matrix handles these. AHB-AP transactions are not subject to MPU lookups. AHB-AP transactions bypass the FPB, and so the FPB cannot remap AHB-AP transactions.

SWJ/SW-DP initiated transaction aborts drive an AHB-AP supported sideband signal called **HABORT**. This signal is driven into the Bus Matrix, which resets the Bus Matrix state, so that AHB-AP can access the Private Peripheral Bus for last ditch debugging such as read/stop/reset the core.

AHB-AP transactions are little endian.

11.7.2 Summary and description of the AHB-AP registers

Table 11-28 lists the AHB-AP registers.

————— Note —————

You can configure any of the AHB-AP registers to be present or not present. Any register that is configured as not present reads as zero.

Table 11-28 AHB-AP register summary

Name	Type	Address	Reset value	Description
Control and Status Word	Read/write	0x00	See Register	See <i>AHB-AP Control and Status Word Register</i> on page 11-40
Transfer Address	Read/write	0x04	-	See <i>AHB-AP Transfer Address Register</i> on page 11-42
Data Read/write	Read/write	0x0C	-	See <i>AHB-AP Data Read/Write Register</i> on page 11-43
Banked Data 0	Read/write	0x10	-	See <i>AHB-AP Banked Data Registers 0-3</i> on page 11-43
Banked Data 1	Read/write	0x14	-	See <i>AHB-AP Banked Data Registers 0-3</i> on page 11-43
Banked Data 2	Read/write	0x18	-	See <i>AHB-AP Banked Data Registers 0-3</i> on page 11-43
Banked Data 3	Read/write	0x1C	-	See <i>AHB-AP Banked Data Registers 0-3</i> on page 11-43
Debug ROM Address	Read only	0xF8	0xE00FF003	See <i>AHB-AP Debug ROM Address Register</i> on page 11-43
Identification Register	Read only	0xFC	0x24770011	See <i>AHB-AP ID Register</i> on page 11-44

AHB-AP Control and Status Word Register

Use this register to configure and control transfers through the AHB interface.

Figure 11-19 shows the bit assignments of the AHB-AP Control and Status Word Register.

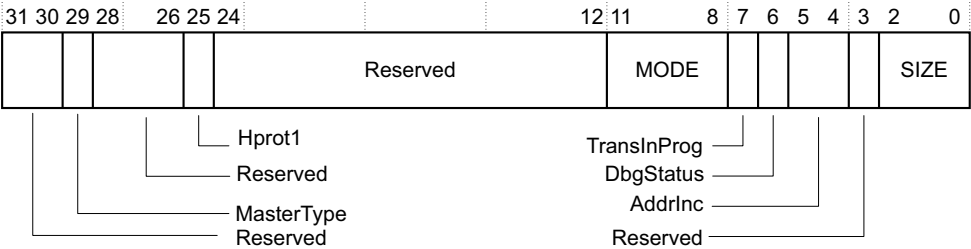


Figure 11-19 AHB-AP Control and Status Word Register

Table 11-29 describes the bit assignments of the AHB-AP Control and Status Word Register.

Table 11-29 AHB-AP Control and Status Word Register bit assignments

Bits	Field	Function
[31:30]	-	Reserved. Read as 0b00.
[29]	MasterType ^a	0 = core. 1 = debug. It cannot be changed if transaction is outstanding. Debugger must first check TransinProg. Reset value = 0b1. If the FIXHMASTERTYPE input signal is set to 1 then this register has no affect on the master value indicated by the transaction. It is always marked as the debugger.
[28:26]	-	Reserved, 0b000.
[25]	Hprot1	User/Privilege control - HPROT[1]. Reset value = 0b1.
[24]	-	Reserved, 0b1.
[23:12]	-	Reserved, 0x000.
[11:8]	Mode	Mode of operation bits: b0000 = normal download/upload mode b0001-b1111 are reserved. Reset value = 0b0000.
[7]	TransINProg	Transfer in progress. This field indicates if a transfer is in progress on the APB master port.
[6]	DbgStatus	Indicates the status of the DAPEN port. If DbgStatus is LOW, no AHB transfers carried out. 1 = AHB transfers permitted. 0 = AHB transfers not permitted.

Table 11-29 AHB-AP Control and Status Word Register bit assignments (continued)

Bits	Field	Function
[5:4]	AddrInc	Auto address increment and pack mode on Read or Write data access. Only increments if the current transaction completes with no error. Auto address incrementing and packed transfers are not performed on access to Banked Data registers 0x10 - 0x1C. The status of these bits is ignored in these cases. Increments and wraps within a 4-KB address boundary, for example for word incrementing from 0x1000 to 0x1FFC. If the start is at 0x14A0, then the counter increments to 0x1FFC, wraps to 0x1000, then continues incrementing to 0x149C. 0b00 = auto increment off. 0b01 = increment single. Single transfer from corresponding byte lane. 0b10 = increment packed. 0b11 = reserved. No transfer. Size of address increment is defined by the Size field [2:0]. Reset value: 0b00.
[3]	-	Reserved.
[2:0]	SIZE	Size of access field: b000 = 8 bits b001 = 16 bits b010 = 32 bits b011-111 are reserved. Reset value: b000.

- a. When clear, this bit prevents the debugger from setting the C_DEBUGEN bit in the Debug Halting Control and Status Register, and so prevent the debugger from being able to halt the core.

AHB-AP Transfer Address Register

Use this register to program the address of the current transfer.

Table 11-30 describes the bit assignments of the AHB-AP Transfer Address Register.

Table 11-30 AHB-AP Transfer Address Register bit assignments

Bits	Field	Function
[31:0]	ADDRESS	Current transfer address. No reset value.

AHB-AP Data Read/Write Register

Use this register to read and write data for the current transfer.

Table 11-31 describes the bit assignments of the AHB-AP Data Read/Write Register.

Table 11-31 AHB-AP Data Read/Write Register bit assignments

Bits	Field	Function
[31:0]	DATA	Write mode: data value to write for the current transfer Read mode: data value to read for the current transfer No reset value.

AHB-AP Banked Data Registers 0-3

Use these registers to directly map AHB-AP accesses to AHB transfers without rewriting the AHB-AP *Transfer Address Register* (TAR).

Table 11-32 describes the field of the AHB-AP Banked Data Registers.

Table 11-32 AHB-AP Banked Data Register bit assignments

Bits	Field	Function
[31:0]	DATA	BD0-BD3 provide a mechanism for directly mapping through DAP accesses to AHB transfers without having to rewrite the TAR within a four location boundary, so for example BD0 reads/write from TAR, BD1 from TAR+4. If DAPADDR[7:4] == 0x0001 , so accessing AHB-AP registers in the range 0x10-0x1C, then the derived HADDR[31:0] is as follows: Read mode: Data value read from the current transfer from external address TAR[31:4] + DAPADDR[3:0] . Auto address incrementing is not performed on DAP accesses to BD0-BD3. Write mode: data value to write for the current transfer to external address TAR[31:4] + DAPADDR[3:0] . Banked transfers are only supported for word transfers. Non-word banked transfer size is currently ignored, assumed word access. No reset value.

AHB-AP Debug ROM Address Register

This register specifies the base address of the debug interface. It is read-only.

Table 11-33 describes the bit assignments of the AHB-AP Debug ROM Address Register.

Table 11-33 AHB-AP Debug ROM Address Register bit assignments

Bits	Field	Function
[31:0]	Debug ROM address	Base address of debug interface.

AHB-AP ID Register

This register defines the external interface on the access port.

Figure 11-20 shows the bit assignments of the AHB-AP ID Register.

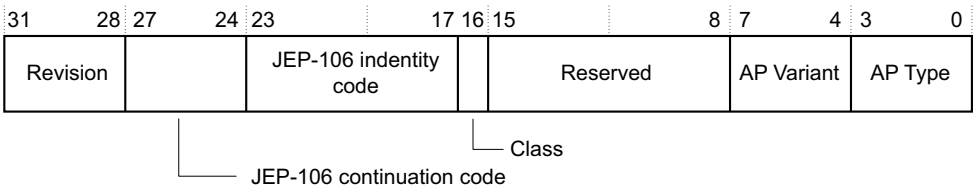


Figure 11-20 AHB-AP ID Register

Table 11-34 describes the bit assignments of the AHB-AP ID Register.

Table 11-34 AHB-AP ID Register bit assignments

Bits	Field	Function
[31:28]	Revision	This field is zero for the first implementation of an AP design, and is updated for each major revision of the design.
[27:24]	JEP-106 continuation code	For an ARM-designed AP, this field has value 0b0100, 0x4.
[23:17]	JEP-106 identity code	For an ARM-designed AP, this field has value 0b0111011, 0x3B.
[16]	Class	0b1: This AP is a Memory Access Port
[15:8]	-	Reserved. SBZ.
[7:4]	AP Variant	0x1: Cortex-M3 variant
[3:0]	AP Type	0x1: AMBA AHB bus

Chapter 12

Bus Interface

This chapter describes the processor bus interface. It contains the following sections:

- *About bus interfaces* on page 12-2
- *AMBA 3 compliance* on page 12-3
- *ICode bus interface* on page 12-4
- *DCode bus interface* on page 12-6
- *System interface* on page 12-7
- *Unifying the code buses* on page 12-9
- *External private peripheral interface* on page 12-10
- *Access alignment* on page 12-11
- *Unaligned accesses that cross regions* on page 12-12
- *Bit-band accesses* on page 12-13
- *Write buffer* on page 12-14
- *Memory attributes* on page 12-15
- *AHB timing characteristics* on page 12-16.

12.1 About bus interfaces

The processor contains four bus interfaces:

- The ICode memory interface. Instruction fetches from Code memory space (0x00000000 - 0x1FFFFFFF) are performed over this 32-bit *Advanced High-performance Bus* (AHB)-Lite bus. For more information, see *ICode bus interface* on page 12-4.
- The DCode memory interface. Data and debug accesses to Code memory space (0x00000000 - 0x1FFFFFFF) are performed over this 32-bit AHB-Lite bus. For more information, see *DCode bus interface* on page 12-6.
- The System interface. Instruction fetches, and data and debug accesses, to System space (0x20000000 - 0xDFFFFFFF, 0xE0100000 - 0xFFFFFFFF) are performed over this 32-bit AHB-Lite bus. For more information, see *System interface* on page 12-7.
- The External *Private Peripheral Bus* (PPB). Data and debug accesses to External PPB space (0xE0040000 - 0xE00FFFFF) are performed over this 32-bit *Advanced Peripheral Bus* (APB) (AMBA v3.0) bus. The *Trace Port Interface Unit* (TPIU) and vendor specific peripherals are on this bus. For more information, see *External private peripheral interface* on page 12-10.

Note

The processor contains an internal PPB for accesses to the *Nested Vectored Interrupt Controller* (NVIC), *Data Watchpoint and Trace* (DWT), *Instrumentation Trace Macrocell* (ITM), *Flash Patch and Breakpoint* (FPB), and *Memory Protection Unit* (MPU).

12.2 AMBA 3 compliance

The processor matches the AMBA 3 specification except for maintaining control information during waited transfers. AMBA 3 AHB-Lite Protocol states that when the slave is requesting wait states the master must not change the transfer type, except for the following cases:

- IDLE transfer - the master is permitted to change the transfer type from IDLE to NONSEQ.
- BUSY transfer, fixed length burst - the master is permitted to change the transfer type from BUSY to SEQ.
- BUSY transfer, undefined length burst - the master is permitted to change from BUSY to any other transfer type.

The processor does not match the given definition because it might change the access type from SEQ or NONSEQ to IDLE during a waited transfer. In effect this cancels the outstanding transfer that has not yet occurred because the previous access is wait-stated and awaiting completion. This enables the processor to have a lower interrupt latency and higher performance in wait-stated systems.

———— **Note** ————

Logic can be implemented external to Cortex-M3 if necessary to achieve total compliance, but this is only required if peripherals require the control information to be maintained through a waited transfer. One way of implementing this is to mask the control information, such as **HTRANS**, while **HREADY** is low.

—————

12.3 ICode bus interface

The ICode interface is a 32-bit AHB-Lite bus interface. Instruction fetches and vector fetches from Code memory space (0x00000000 - 0x1FFFFFFF) are performed over this bus.

Only the CM3Core instruction fetch bus can access the ICode interface, enabling optimal code fetch performance. All fetches are word wide. The number of instructions fetched per word depends on the code running and the alignment of the code in memory. Table 12-1 describes this.

————— **Note** —————

It is strongly recommended that any external arbitration between the and DCode AHB bus interfaces ensures that DCode has a higher priority than ICode.

Table 12-1 Instruction fetches

32-bit instruction fetch [31:16]	32-bit instruction fetch [15:0]	Description
Thumb16[15:0]	Thumb16[15:0]	All Thumb instructions are halfword aligned in memory, so two 16-bit Thumb instructions are fetched at a time. For sequential code, an instruction fetch is performed every second cycle. Instruction fetches can be performed on back-to-back cycles if there is an interrupt or a branch.
Thumb32[31:16]	Thumb32[15:0]	If 32-bit Thumb instructions are word-aligned in memory, then a complete 32-bit Thumb instruction is fetched each cycle.
Thumb32[15:0]	Thumb32[31:16]	If 32-bit Thumb instructions are halfword aligned, then the first 32-bit fetch only returns the first halfword of the 32-bit Thumb instruction. A second fetch must be performed to fetch the second halfword. This scenario creates a wait cycle (a cycle where CM3Core is not able to execute an instruction) depending on the instruction in play. The additional cycle of latency only occurs for the first halfword aligned 32-bit Thumb instruction fetch. CM3Core contains a 3-entry fetch buffer, and so the upper halfword of halfword aligned 32-bit Thumb instructions exist in the fetch buffer for subsequent sequential 32-bit Thumb instructions.

All ICode instruction fetches are marked as cacheable and non-bufferable, **HPROTI[3:2]** = 2'b10, and as allocate and non-shareable, **MEMATTRI** = 2'b01. These attributes are hard wired. If an MPU is fitted, the MPU region attributes are ignored for the ICode bus.

HPROTI[0] indicates what is being fetched:

- 0 - instruction fetch
- 1 - vector fetch.

All ICode transactions are performed as non-sequentials.

12.3.1 Branch status signal

A branch status signal, **BRCHSTAT**, is exported on the *Embedded Trace Macrocell* (ETM) interface that indicates if there are any branches in the pipeline. A prefetcher, for example, can use this to prevent prefetching if a branch is about to be fetched. For more information about the branch status signal, see Chapter 15 *Embedded Trace Macrocell Interface*.

12.4 DCode bus interface

The DCode interface is a 32-bit AHB-Lite bus. Data and debug accesses to Code memory space (0x00000000 - 0x1FFFFFFF) are performed over this bus. Core data accesses have a higher priority than debug accesses. This means that debug accesses are waited until core accesses have completed when there are simultaneous core and debug access to this bus.

Control logic in this interface converts unaligned data and debug accesses into two or three (depending on the size and alignment of the unaligned access) aligned accesses. This stalls any subsequent data or debug access until the unaligned access has completed.

See *Access alignment* on page 12-11 for a description of unaligned accesses.

———— Note ————

It is strongly recommended that any external arbitration between the ICode and DCode AHB bus interfaces ensures that DCode has a higher priority than ICode.

12.4.1 Exclusives

The DCode bus supports exclusive accesses. This is carried out using two sideband signals, **EXREQD** and **EXRESPD**. For more information, see *DCode interface* on page A-9.

For more information about semaphores and the local exclusive monitor see the ARM Architecture Memory Model chapter in the *ARMv7M ARM Architecture Reference Manual*.

12.4.2 Memory attributes

All DCode memory accesses are marked as cacheable and non-bufferable, **HPROTD[3:2] = 2'b10**, and as allocate and non-shareable, **MEMATTRD = 2'b01**.

These attributes are hard wired. If an MPU is fitted, the MPU region attributes are ignored for the DCode bus.

12.5 System interface

The system interface is a 32-bit AHB-Lite bus. Instruction and vector fetches, and data and debug accesses to the System memory space, 0x20000000 - 0xDFFFFFFF, 0xE0100000 - 0xFFFFFFFF, are performed over this bus.

For simultaneous accesses to this bus, the arbitration order in decreasing priority is:

- data accesses
- instruction and vector fetches
- debug.

The System bus interface contains control logic to handle unaligned accesses, FPB remapped accesses, bit-band accesses, and pipelined instruction fetches.

12.5.1 Unaligned accesses

Unaligned data and debug accesses are converted into two or three aligned accesses, depending on the size and alignment of the unaligned access. This stalls any subsequent accesses until the unaligned access has completed. For a description of unaligned accesses, see *Access alignment* on page 12-11.

12.5.2 Bit-band accesses

Accesses to the bit-band alias region are converted into accesses to the bit-band region. Bit-band writes take two cycles, they are converted into read-modify-write operations, and so bit-band write accesses stall any subsequent accesses until the bit-band access has completed. For a description of bit-band accesses, see *Bit-band accesses* on page 12-13.

12.5.3 Flash Patch remapping

Accesses to the Code memory space that are remapped to System memory space incur a cycle penalty to be remapped. This stalls any subsequent accesses until the Flash Patch access has completed. See *FPB* on page 11-6 for a description of Flash Patch.

12.5.4 Exclusives

The System bus supports exclusive accesses. This is carried out using two sideband signals, **EXREQS** and **EXRESPS**. For more information, see *System bus interface* on page A-10.

For more information about semaphores and the local exclusive monitor see the ARM Architecture Memory Model chapter in the *ARMv7M ARM Architecture Reference Manual*.

12.5.5 Memory attributes

The processor exports memory attributes on the System bus by using a sideband bus called **MEMATTRS**. For more information, see *Memory attributes* on page 12-15.

12.5.6 Pipelined instruction fetches

To provide a clean timing interface on the System bus, instruction and vector fetch requests to this bus are registered. This results in an additional cycle of latency because instructions fetched from the System bus take two cycles. This also means that back-to-back instruction fetches from the System bus are not possible.

———— **Note** —————

Instruction fetch requests to the ICode bus are not registered. Performance critical code must run from the ICode interface.

—————

12.6 Unifying the code buses

For some systems you might want to combine the processor core's ICode and DCode buses into a single, unified Code bus. To support this for high-speed operation, the processor has the **DNOTITRANS** input that suppresses the **HTRANSI** line when **HTRANSD** becomes active. With **DNOTITRANS** asserted, if **HTRANSI** and **HTRANSD** are to be active simultaneously in corresponding single-cycle address phases, then only **HTRANSD** is asserted. The ICode transaction is waited internal to the processor. In other words, the external ICode bus is forced into an idle state. The two **HTRANS** signals are therefore guaranteed never to be simultaneously active, which permits the bus multiplexer to be a very simple device.

Note

DNOTITRANS is a static input that must be tied high to enforce this behavior.

The external ICode/DCode bus multiplexer can be integrated into a Cortex-M3 system as Figure 12-1 shows.

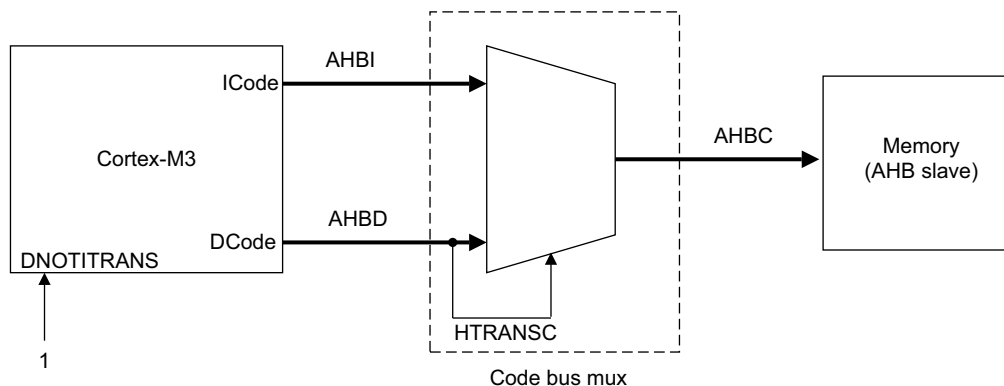


Figure 12-1 ICode/DCode multiplexer

12.7 External private peripheral interface

The external private peripheral interface is an APB (AMBA v3.0) bus. Data and debug accesses to the External Peripheral memory space (0xE0040000 - 0xE00FFFFF) are performed over this bus. The TPIU and any vendor specific components populate this bus. Core data accesses have higher priority than debug accesses, so debug accesses are waited until core accesses have completed when there are simultaneous core and debug access to this bus. Only the address bits necessary to decode the External PPB space are supported on this interface. These address bits are bits [19:2] of **PADDR**.

PADDR31 is driven as a sideband signal on this bus. When the signal is HIGH, it indicates that the AHB-AP debug is the requesting master. When the signal is LOW, it indicates that the core is the requesting master.

Unaligned accesses to this bus are architecturally Unpredictable and are not supported. The processor drives out the original **HADDR[1:0]** request from the core and does not convert the request into multiple aligned accesses.

12.8 Access alignment

The processor supports unaligned data accesses using the ARMv6 model. The DCode and System bus interfaces contain logic that converts unaligned accesses to aligned accesses.

The unaligned data accesses are described in Table 12-2. The table shows the unaligned access in the first column, with the remaining columns showing what the access is converted into. Depending on the size and alignment of the unaligned access, it is converted into two or three aligned accesses.

Table 12-2 Bus mapper unaligned accesses

Unaligned access		Aligned access					
		Cycle 1		Cycle 2		Cycle 3	
Size	ADDR [1:0]	HSIZE	HADDR [1:0]	HSIZE	HADDR[1:0]	HSIZE	HADDR[1:0]
Halfword	00	Halfword	00	-	-	-	-
Halfword	01	Byte	01	Byte	10	-	-
Halfword	10	Halfword	10	-	-	-	-
Halfword	11	Byte	11	Byte	{(Addr+4)[31:2],2b00}	-	-
Word	00	Word	00	-	-	-	-
Word	01	Byte	01	Halfword	10	Byte	{(Addr+4)[31:2],2b00}
Word	10	Halfword	10	Halfword	{(Addr+4)[31:2],2b00}	-	-
Word	11	Byte	11	Halfword	{(Addr+4)[31:2],2b00}	Byte	{(Addr+4)[31:2],2b10}

Note

Unaligned accesses that cross into the bit-band alias region are not treated as bit-band requests, and the access is not remapped to the bit-band region. Instead, they are treated as a halfword or byte access to the bit-band alias region.

12.9 Unaligned accesses that cross regions

The CM3Core supports ARMv6 unaligned accesses, and performs all accesses as single, unaligned accesses. They are converted into two or more aligned accesses by the DCode and System bus interfaces.

Note

All Cortex-M3 external accesses are aligned.

Unaligned support is only available for load/store singles (LDR, STR). Load/store double already supports word aligned accesses, but does not permit other unaligned accesses, and generates a fault if this is attempted.

Unaligned accesses that cross memory map boundaries are architecturally Unpredictable. The processor behavior is boundary dependent, as follows:

- DCode accesses wrap within the region. For example, an unaligned halfword access to the last byte of Code space (0x1FFFFFFF) is converted by the DCode interface into a byte access to 0x1FFFFFFF followed by a byte access to 0x00000000.
- System accesses that cross into PPB space do not wrap within System space. For example, an unaligned halfword access to the last byte of System space (0xDFFFFFFF) is converted by the System interface into a byte access to 0xDFFFFFFF followed by a byte access to 0xE0000000. 0xE0000000 is not a valid address on the System bus.
- System accesses that cross into Code space do not wrap within System space. For example, an unaligned halfword access to the last byte of System space (0xFFFFFFFF) is converted by the System interface into a byte access to 0xFFFFFFFF followed by a byte access to 0x00000000. 0x00000000 is not a valid address on the System bus.
- Unaligned accesses are not supported to PPB space, and so there are no boundary crossing cases for PPB accesses.

Unaligned accesses that cross into the bit-band alias regions are also architecturally Unpredictable. The processor performs the access to the bit-band alias address, but this does not result in a bit-band operation. For example, an unaligned halfword access to 0x21FFFFFF is performed as a byte access to 0x21FFFFFF followed by a byte access to 0x22000000 (the first byte of the bit-band alias).

Unaligned loads that match against a literal comparator in the FPB are not remapped. FPB only remaps aligned addresses.

12.10 Bit-band accesses

The System bus interface contains logic that controls bit-band accesses as follows:

- It remaps bit-band alias addresses to the bit-band region.
- For reads, it extracts the requested bit from the read byte, and returns this in the *Least Significant Bit* (LSB) of the read data returned to the core.
- For writes, it converts the write to an atomic read-modify-write operation.

For more information about bit-banding, see *Bit-banding* on page 4-5.

Note

- The Cortex-M3 core does not stall during bit-band operations unless it attempts to access the System bus while the bit-band operation is being carried out.
-

12.11 Write buffer

To prevent bus wait cycles from stalling the processor during data stores, buffered stores to the DCode and System buses go through a one-entry write buffer. If the write buffer is full, subsequent accesses to the bus stall until the write buffer has drained. The write buffer is only used if the bus waits the data phase of the buffered store, otherwise the transaction completes on the bus.

DMB and DSB instructions wait for the write buffer to drain before completing. If an interrupt comes in while DMB/DSB is waiting for the write buffer to drain, the opcode after the DMB/DSB is returned to on the completion of the interrupt. This is because interrupt processing is a memory barrier operation.

12.12 Memory attributes

The processor exports memory attributes on the System bus by the addition of a sideband bus, **MEMATTR**.

Table 12-3 shows the relationship between **MEMATTR[0]** and **HPROT[3:2]**.

Table 12-3 Memory attributes

MEMATTR[0]	HPROT[3]	HPROT[2]	Description
0	0	0	Strongly ordered
0	0	1	Device
0	1	0	L1 cacheable, L2 not cacheable
1	0	0	Invalid
1	0	1	Invalid
1	1	0	Cache WT, allocate on read
0	1	1	Cache WB, allocate on read and write
1	1	1	Cache WB, allocate on read

12.13 AHB timing characteristics

The processor does not contain memories within the macrocell. To achieve high system performance, and to give the implementor complete flexibility in their memory architecture, memory requests from the processor are presented directly to the AHB interfaces unregistered.

Because of this, the Cortex-M3 AHB outputs are valid approximately 50% into the cycle, and the AHB inputs have a setup requirement of approximately 50% of the clock period.

Table 12-4 describes the timing characteristics of each of the interfaces.

Table 12-4 Interface timing characteristics

interface	Timing characteristics
ICODE	Instruction address and control signals are generated from the ALU, and as a result are valid approximately 50% into the clock cycle. Read data (HRDATAI) and read response (HRESPI) are presented directly to the processor and have approximately 50% of clock period setup.
DCODE	Core data and debug requests are presented over this bus. Both data and debug requests are presented relatively early in the cycle, and they are generated from registers with a small amount of combinatorial logic after the register. Requests on this bus have more slack than those presented on the ICODE bus. Write data (HWDATAD) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (HRDATAD) and read response (HRESPD) are presented directly to the processor and have approximately 50% of clock period setup.
SYSTEM	Instruction fetches from this bus are pipelined, as described in <i>Pipelined instruction fetches</i> on page 12-8, and data and debug requests to this bus are presented early in the cycle, so requests on this bus have more slack than those presented on the ICODE bus. Write data (HWDATAS) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (HRDATAS) and read response (HRESPS) are presented directly to the processor and have approximately 50% of clock period setup.
PPB	Data and debug requests to this bus are presented early in the cycle, so requests on this bus have more slack than those presented on the ICODE bus. Write data (PWDATA) is presented directly from the ALU and is valid approximately 50% into the clock cycle. Read data (PRDATA) is presented directly to the processor and has approximately 50% of clock period setup

Chapter 13

Debug Port

This chapter describes the processor *Debug Port* (DP). It contains:

- *About the DP* on page 13-2.

13.1 About the DP

The processor contains an *Advanced High-performance Bus Access Port* (AHB-AP) interface for debug accesses. An external DP component accesses this interface. The Cortex-M3 system supports three possible DP implementations:

- The *Serial Wire JTAG Debug Port* (SWJ-DP). The SWJ-DP is a standard CoreSight debug port that combines JTAG-DP and *Serial Wire Debug Port* (SW-DP).
- The SW-DP. This provides a two-pin (clock + data) interface to the AHB-AP port.
- No DP present. If no debug functionality is present within the processor, a DP is not required.

Note

The SWJ-DP is designed to permit pin sharing of **JTAG-TDO** and **JTAG-TDI** when they are not being used for JTAG debug access. When used together with a Cortex-M3 TPIU, there are different options for the connection of *Serial Wire Output* (SWO), see *Serial wire output connection* on page 17-21.

The two DP implementations provide different mechanisms for debug access to the processor. Your implementation must contain only one of these components.

Note

Your implementation might contain an alternative implementor-specific DP instead of SW-DP or SWJ-DP. See your implementor for details.

For more detailed information on the DP components, see the *CoreSight Components Technical Reference manual*.

For more information on the AHB-AP, see *AHB-AP* on page 11-39.

The DP and AP together are referred to as the *Debug Access Port* (DAP).

For more detailed information on the debug interface, see the *ARM Debug Interface v5, Architecture Specification*.

Chapter 14

Embedded Trace Macrocell

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

- *About the ETM* on page 14-2
- *Data tracing* on page 14-7
- *ETM resources* on page 14-8
- *Trace output* on page 14-11
- *ETM architecture* on page 14-12
- *ETM programmer's model* on page 14-16.

14.1 About the ETM

The ETM is an optional debug component that enables reconstruction of program execution. The ETM is designed to be a high-speed, low-power debug tool that only supports instruction trace. This ensures that area is minimized, and that gate count is reduced.

14.1.1 ETM block diagram

Figure 14-1 on page 14-3 shows a block diagram of the ETM, and shows how the ETM interfaces to the *Trace Port Interface Unit* (TPIU).

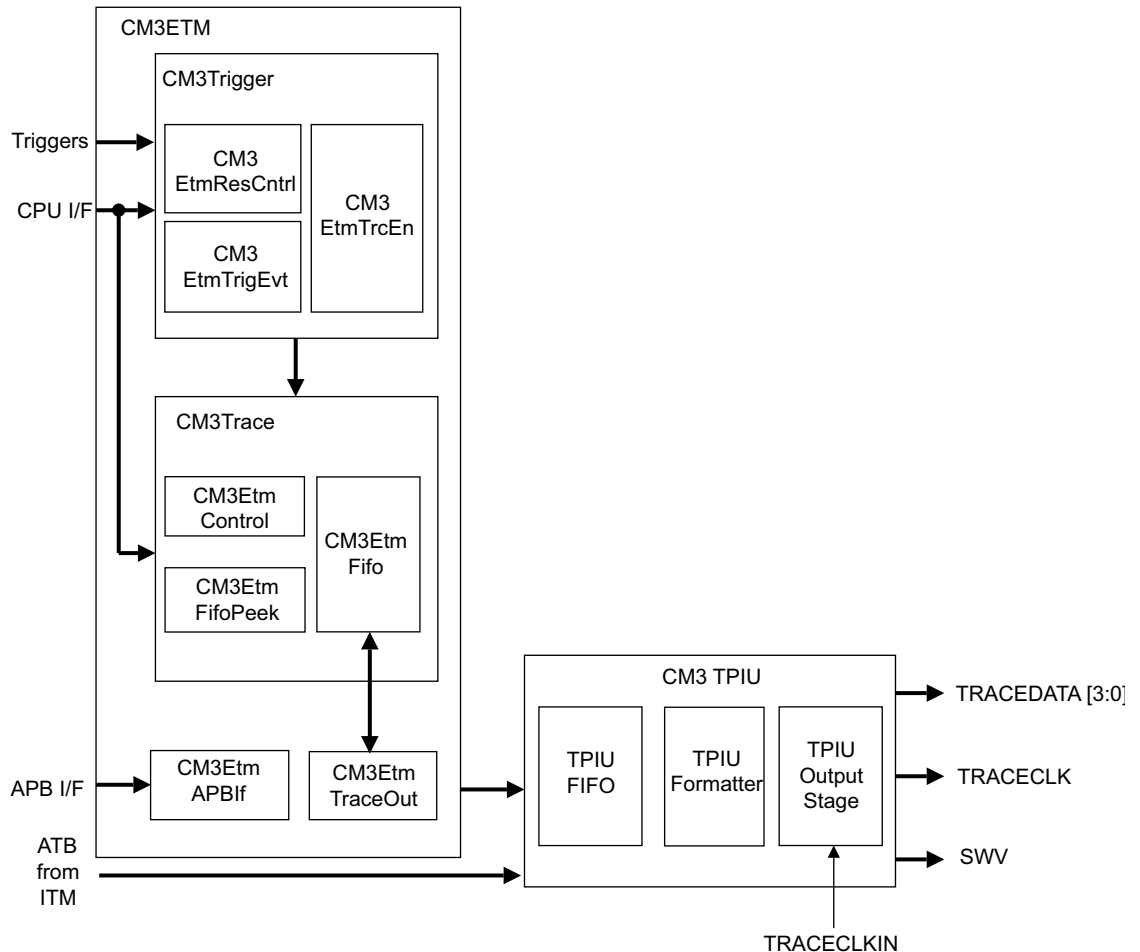


Figure 14-1 ETM block diagram

14.1.2 ETM inputs and outputs

This section describes the ETM inputs and outputs:

- ETM core interface. See Table 14-1 on page 14-4.
- Miscellaneous configuration inputs. See Table 14-2 on page 14-4.
- Trace port signals. See Table 14-2 on page 14-4.
- Other signals. See Table 14-4 on page 14-5.
- Clocks and resets. See Table 14-5 on page 14-6.

- *Advanced Peripheral Bus* (APB) interface signals. See Table 14-6 on page 14-6.

Table 14-1 ETM core interface inputs and outputs

Name	Description	Qualified by	Direction
ETMIA[31:1]	Core instruction address bus.	ETMIVALID	Input
ETMIVALID	Current instruction data represents an instruction.	-	Input
ETMDVALID	Current instruction data represents an instruction.	-	Input
ETMICCFAIL	Instruction failed its condition code.	ETMIVALID	Input
ETMIBRANCH	Instruction is a branch target.	ETMIVALID	Input
ETMIINDBR	Instruction is an indirect branch target.	ETMIBRANCH	Input
ETMFLUSH	PC modified before next instruction.	-	Input
ETMISTALL	Indicates that the last instruction signalled by the core has not yet entered execute.	-	Input
ETMFINDBR	PC modified by an indirect operation.	ETMFLUSH	Input
ETMINTSTAT[2:0]	Exception entry and exit.	-	Input
ETMINTNUM[8:0]	Exception type.	ETMINTSTAT	Input
ETMCANCEL	Exception is a canceling exception.	ETMINTSTAT	Input
COREHALT	Core is halted.	-	Input
DWTMATCH [3:0]	Indicates that the <i>Data Watchpoint and Trace</i> (DWT) trigger units have matched the conditions currently present on the address, data and control buses.	-	Input
DWTINOTD[3:0]	Indicates that the DWT trigger units are performing comparisons on PC value (set) or data address (clear).	-	Input

Table 14-2 Miscellaneous configuration inputs

Name	Description	Direction	Clock domain
NIDEN	Non invasive debug enable	Input	FCLK
EXTIN[1:0]	External input resource	Input	FCLK

Table 14-2 Miscellaneous configuration inputs (continued)

Name	Description	Direction	Clock domain
MAXEXTIN[1:0]	Maximum supported external inputs	Input	FCLK
CGBYPASS	Bypass architectural clock gating cell	Input	FCLK
FIFOFULLEN	Enable ETMFIFOFULL	Input	FCLK

Note

One of the **EXTIN** inputs to the ETM could be driven from the **LOCKUP** output from the core to enable trace capture to stop, or trigger if a lockup condition occurs. The **EXTIN** inputs are not synchronized in the ETM. If they are not driven from the ETM clock, then you must synchronize them outside the ETM.

Table 14-3 Trace port signals

Name	Description	Direction	Clock domain
ATDATAM[7:0]	Eight-bit trace data	Output	FCLK
ATVALIDM	ATDATA is valid	Output	FCLK
ATIDM[6:0]	Trace Source ID	Output	FCLK
ATREADYM	Indicates that the Trace Port is able to accept the Data on ATDATA	Input	FCLK
AFREADYM	Indicates that the ETM FIFO is empty	Output	FCLK

Table 14-4 Other signals

Name	Description	Direction	Clock domain
FIFOPEEK[9:0]	For validation purposes only	Output	FCLK
FIFOFULL	ETM FIFO is full	Output	FCLK
ETMPWRUP	Indicates that the ETM is powered up	Output	FCLK
ETMTRIGOUT	Trigger occurred status signal	Output	HCLK
ETMDBGRQ	Debug request to core	Output	FCLK
ETMEN	ETM traceport enabled	Output	FCLK

Table 14-5 Clocks and resets

Name	Description	Direction
FCLK	Clock for ETM logic which must be connected to the same FCLK as Cortex-M3.	Input
PORRESETn	Power on reset for the HCLK domain. Must not be the same as core HCLK reset (SYSRESETn).	Input

Table 14-6 APB interface signals

Name	Description	Direction	Clock domain
PSEL	APB device select	Input	FCLK
PENABLE	APB control signal	Input	FCLK
PADDR[11:2]	APB Address Bus	Input	FCLK
PWRITE	APB Transfer direction (!Read/Write)	Input	FCLK
PWDATA[31:0]	APB Write Data Bus	Input	FCLK
PRDATA[31:0]	APB Read Data Bus	Output	FCLK

14.2 Data tracing

The Cortex-M3 system can perform low-bandwidth data tracing using the *Data Watchpoint and Trace* (DWT) and *Instruction Trace Macrocell* (ITM) components. To enable support of instruction trace with a low pin-count, data trace is not included in the ETM. This considerably reduces gate count for the ETM, because the triggering resources are simplified.

When the ETM is implemented in the processor, the two trace sources, ITM and ETM, both feed into the TPIU, where they are combined and usually output over the trace port. DWT is able to provide either focused data trace, or global data trace, subject to FIFO overflow issues. The TPIU is optimized for the requirements of a single core Cortex-M3 system.

14.3 ETM resources

Because the ETM does not generate data trace information, the lower bandwidth reduces the requirement for complex triggering capabilities. This means that the ETM does not include the following:

- internal comparators
- counters
- sequencers.

Table 14-7 lists the Cortex-M3 resources.

Table 14-7 Cortex-M3 resources

Feature	Present on Cortex-M3 ETM
Architecture version	ETMv3.4
Address comparator pairs	0
Data comparators	0
Context ID comparators	0
MMDs	0
Counters	0
Sequencer	No
Start/stop block	Yes
Embedded ICE comparators	4
External inputs	2
External outputs	0
Extended external inputs	0
Extended external input selectors	0
FIFOFULL	Yes
FIFOFULL level setting	Yes
Branch broadcasting	Yes
ASIC Control Register	No
Data suppression	No

Table 14-7 Cortex-M3 resources (continued)

Feature	Present on Cortex-M3 ETM
Software access to registers	Yes
Readable registers	Yes
FIFO size	24 bytes
Minimum port size	8 bits
Maximum port size	8 bits
Normal port mode	-
Normal half-rate clocking/1:1	Yes - asynchronous
Demux port mode	-
Demux half-rate clocking/1:2	No
Mux port mode/2:1	No
1:4 port mode	No
Dynamic port mode (including stalling)	No. Supported by asynchronous port mode.
CPRT data	No
Load PC first	No
Fetch comparisons	No
Load data traced	No

14.3.1 Periodic synchronization

The ETM uses a fixed synchronization packet generation frequency of every 1024 bytes of trace.

14.3.2 Data and instruction address compare resources

The DWT provides four address comparators on the data bus that provide debug functionality. Within the DWT unit, it is possible to specify the functions triggered by a match, and one of these functions is to generate an ETM match input. These inputs are presented to the ETM as *Embedded In Circuit Emulator (ICE)* comparator inputs.

A single DWT resource can trigger an ETM event and also generate instrumentation trace directly from the same event.

You can also individually configure the four DWT comparators to compare with the execute PC to permit the ETM access to a PC compare resource. These inputs are presented to the ETM as Embedded ICE comparator inputs.

Note

Using a DWT comparator as a PC comparator reduces the number of available data address comparisons.

See *DWT* on page 11-13 for more information about the DWT unit.

External inputs

Two external inputs, **ETMEXTIN[1:0]**, enable additional on-chip IP to generate trigger/enable signals for the ETM.

Start/stop block

The start/stop block controls start/stop behavior by using the embedded ICE inputs to the ETM. The DWT controls these inputs.

14.3.3 FIFO functionality

The FIFO size is 24 bytes.

A FIFOFULL output is provided to enable the core to be stalled when the FIFO reaches a specific depth. Although stalling the core in a typical application is unlikely to be acceptable, it provides a mechanism for enabling 100% trace that could be compared with the partial trace obtained for a non-stalled run.

14.4 Trace output

The ETM outputs data 8 bits at a time, at the core clock speed. It does not support different trace port sizes and trace port modes. The TPIU exports trace output off chip. This output is compatible with the *AMBA Trace Bus* (ATB) protocol.

Because AFVALID functionality is not supported, the trace port cannot flush data from the ETM FIFO. However, with an 8-bit ATB port the FIFO always drains, which makes AFVALID unnecessary.

The Cortex-M3 system is equipped with an optimized TPIU that is designed for use with the ETM and ITM. This TPIU does not support additional trace sources. However, you can add additional trace sources if the TPIU has been replaced with a more complex version, and more trace infrastructure.

Note

A trace ID register and output are provided for systems that use multiple trace sources.

The TPIU uses the formatted trace output protocol. This means that there is no requirement for an extra pin for **TRACECTL** signal.

Trace output from the ETM is synchronous to the core clock. There is an asynchronous FIFO in the trace port interface. If you want to integrate the ETM into a multi-core system, you might have to use an asynchronous ATB bridge.

14.5 ETM architecture

The ETM is an instruction only ETM that implements ARM ETM architecture v3.4. It is based on the ARM ETM Architecture Specification. For full details, see the *ARM Embedded Trace Macrocell Architecture Specification*.

All 32-bit Thumb instructions are traced as a single instruction. Instructions following an IT instruction are traced as normal conditional instructions. The decompressor does not have to refer to the IT instruction.

14.5.1 Restartable instructions

The ARMv7-M architecture can restart LSM instructions that are interrupted by an exception. The ETM traces an instruction that has been interrupted by an exception by indicating that it has been cancelled. On return from the exception, the ETM traces the same instruction again, regardless of the instruction being restarted or resumed.

14.5.2 Exception return

The ETM explicitly indicates return from an exception in the trace stream. This is because exception return functionality is encoded in a data-dependent manner, and an exception return behaves differently from a simple branch.

The packet encoding indicates a return from an exception. Figure 14-2 shows this.

7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0

Figure 14-2 Return from exception packet encoding

If a new, higher priority exception pre-empts the stack pop, the branch to the exception handler must indicate that the last instruction was cancelled. This indicates that the return from exception packet was cancelled, but the return from exception instruction was not cancelled. If the return from exception packet is present, then this means that the previous instruction did complete.

14.5.3 Exception tracing

To trace exceptions, an optional field is added to a branch packet. This extra field specifies the exception information. A normal branch packet is encoded in 1-5 bytes of trace data, while the exception branch is as follows:

- 2-5 bytes of address

- 1-2 bytes of exception.

The exception mapping is designed to enable the most frequent exceptions to be encoded within one byte. The ETM exception tracing mapping is described in Table 14-8.

Table 14-8 Exception tracing mapping

Number of bytes	Exception	ETMINTNUM	Traced value
1 byte exception	None	-	0
1 byte exception	IRQ1	17	1
1 byte exception	IRQ2	18	2
1 byte exception	IRQ3	19	3
1 byte exception	IRQ4	20	4
1 byte exception	IRQ5	21	5
1 byte exception	IRQ6	22	6
1 byte exception	IRQ7	23	7
1 byte exception	IRQ0	16	8
1 byte exception	Usage Fault	6	9
1 byte exception	NMI	2	10
1 byte exception	SVC	11	11
1 byte exception	DebugMon	12	12
1 byte exception	MemManage	4	13
1 byte exception	PendSV	14	14
1 byte exception	SysTick	15	15
2 bytes exception	Reserved	8	16
2 bytes exception	Reset	0	17
2 bytes exception	Reserved	10	18
2 bytes exception	HardFault	3	19
2 bytes exception	Reserved	9	20
2 bytes exception	BusFault	5	21

Table 14-8 Exception tracing mapping (continued)

Number of bytes	Exception	ETMINTNUM	Traced value
2 bytes exception	Reserved	7	22
2 bytes exception	Reserved	13	23
2 bytes exception	IRQ8	24	24
2 bytes exception	IRQ9	25	25
2 bytes exception	IRQ10	26	26
.	.	.	.
.	.	.	.
.	.	.	.
2 bytes exception	IRQ239	255	255

Figure 14-3 shows the full branch with exception packet.

7	6	5	4	3	2	1	0	
C	Addr[6:1]						1	Address byte 0
C	E/ Addr[13]	Addr[12:7]						Address byte 1 (optional)
C	E/ Addr[20]	Addr[19:14]						Address byte 2 (optional)
C	E/ Addr[27]	Addr[26:21]						Address byte 3 (optional)
C	E	0	1	Addr[31:28]				Address byte 4 (optional)
C	T2EE	Canc	Excp[3:0]				NS	Exception information Byte 0
C	0	SBZ	Excp[8:4]					Exception information Byte 1 (optional)

Figure 14-3 Exception encoding for branch packet

The final address byte uses bits [7:6] set to 0b01 to indicate the end of the address field. Exception data follows this field. Exception byte 0 sets bit [7] to 1 if a second exception byte follows. If there is no exception present, and only address bits [6:1] change, then a single byte is used. If an exception is present, then at least two bytes signal the address.

When turning off trace immediately before entry to an exception handler, the ETM remains enabled until the exception is taken. This enables it to trace the branch address, exception type and resume information.

14.6 ETM programmer’s model

The ETM programmer’s model is described in detail in the *ARM Embedded Trace Macrocell Architecture Specification*. This section defines the implementation-specific features of the ETM programmer’s model.

14.6.1 Advanced Peripheral Bus interface

The ETM contains an APB slave interface that can read and write to the ETM registers. This interface is synchronous to the processor clock. The core and the external debug interface can access it through the *Serial Wire Debug Port/JTAG Debug Port* (SW-DP/JTAG-DP).

14.6.2 List of ETM registers

The ETM registers are listed in Table 14-9. For full details, see the *ARM Embedded Trace Macrocell Architecture Specification*.

Table 14-9 ETM registers

Name	Type	Address	Present	Description
ETM Control	Read/write	0xE0041000	Yes	For a description, see page 14-19.
Configuration Code	Read only	0xE0041004	Yes	For a description, see page 14-20.
Trigger Event	Read/write	0xE0041008	Yes	Defines the event that controls the trigger. [16:14] Boolean function. [13:7] Resource A. [6:0] Resource B See <i>ETM Event resources</i> on page 14-22.
ASIC Control	-	0xE004100C	No	-
ETM Status	Read/write	0xE0041010	Yes	Provides information on the current status of the trace and trigger logic. [3] - Trigger Flag. [2] - Start/Stop resource status. [1] - Programming bit status. [0] - Untraced Overflow.
System Configuration	Read only	0xE0041014	Yes	For a description, see page 14-20.

Table 14-9 ETM registers (continued)

Name	Type	Address	Present	Description
Trace Start/Stop Resource Control	-	0xE0041018	No	-
TraceEnable Event	Write only	0xE0041020	Yes	Describes the TraceEnable enabling event. [16:14] Boolean function. [13:7] Resource A. [6:0] Resource B. See <i>ETM Event resources</i> on page 14-22.
TraceEnable Control 1	Write only	0xE0041024	Yes	For a description, see page 14-21.
TraceEnable Control 2	Write only	0xE004101C	No	-
FIFOFULL Region	Write only	0xE0041028	No	If enabled, FifoFull logic is always active.
FIFOFULL Level	Read/write	0xE004102C	Yes	The number of bytes left in the FIFO, below which the FIFOFULL signal is asserted to stall the core. Bit [7] of the ETM Control Register is used to enable the FIFOFULL output.
ViewData	-	0xE0041030- 0xE004103C	No	-
Address Comparators	-	0xE0041040- 0xE004113C	No	-
Counters	-	0xE0041140- 0xE0041157C	No	-
Sequencer	-	0xE0041180- 0xE0041194, 0xE0041198	No	-
External Outputs	-	0xE00411A0- 0xE00411AC	No	-
CID Comparators	-	0xE00411B0- 0xE00411BC	No	-
Implementation specific	-	0xE00411C0- 0xE00411DC	No	All RAZ. Ignore writes.

Table 14-9 ETM registers (continued)

Name	Type	Address	Present	Description
Synchronization Frequency	Read only	0xE00411E0	Yes	Reads as 0x00000400.
ETM ID	Read only	0xE00411E4	Yes	For a description, see page 14-21.
Configuration Code Extension	Read only	0xE00411E8	Yes	For a description, see page 14-21.
Extended External Input Selector	-	0xE00411EC	No	No extended external inputs implemented.
TraceEnable Start/Stop Embedded ICE	Read/write	0xE00411F0	Yes	Bits [19:16] configure DWT comparator inputs to use as stop resources. Bits [3:0] configure DWT Comparator inputs to use as start resources.
Embedded ICE Behavior Control	-	0xE00411F4	No	Embedded ICE (DWT comparator) inputs use the default behavior.
CoreSight Trace ID	Read/write	0xE0041200	Yes	Implemented as normal. Values of 0x00, 0x70–0x7F are reserved and must not be used when the ETM is active.
OS Save/Restore	-	0xE0041304– 0xE0041308	No	OS Save/Restore not implemented. RAZ, ignore writes.
Power Down Status Register	Read only	0xE0041314	Yes	For a description, see page 14-22.
ITMISCIN	Read only	0xE0041EE0	Yes	Sets [1:0] to EXTIN[1:0], [4] to COREHALT.
ITTRIGOUT	Write only	0xE0041EE8	Yes	Sets [0] to TRIGGER.
ITATBCTR2	Read only	0xE0041EF0	Yes	Sets [0] to ATREADY.
ITATBCTR0	Write only	0xE0041EF8	Yes	Sets [0] to ATVALID.
Integration Mode Control	Read/write	0xE0041F00	Yes	Implemented as normal.
Claim Tag	Read/write	0xE0041FA0– 0xE0041FA4	Yes	Implements the 4-bit claim tag.
Lock Access	Write only	0xE0041FB0– 0xE0041FB4	Yes	Implemented as normal.
Lock Status	Read only	0xE0041Fb4	Yes	Implemented as normal.

Table 14-9 ETM registers (continued)

Name	Type	Address	Present	Description
Authentication Status	Read only	0xE0041FB8	Yes	Implemented as normal.
Device Type	Read only	0xE0040FCC	Yes	Reset value: 0x13.
Peripheral ID 4	Read only	0xE0041FD0	Yes	0x04
Peripheral ID 5	Read only	0xE0041FD4	Yes	0x00
Peripheral ID 6	Read only	0xE0041FD8	Yes	0x00
Peripheral ID 7	Read only	0xE0041FDC	Yes	0x00
Peripheral ID 0	Read only	0xE0041FE0	Yes	0x24
Peripheral ID 1	Read only	0xE0041FE4	Yes	0xB9
Peripheral ID 2	Read only	0xE0041FE8	Yes	0x2B
Peripheral ID 3	Read only	0xE0041FEC	Yes	0x00
Component ID 0	Read only	0xE0041FF0	Yes	0x0D
Component ID 1	Read only	0xE0041FF4	Yes	0x90
Component ID 2	Read only	0xE0041FF8	Yes	0x05
Component ID 3	Read only	0xE0041FFC	Yes	0xB1

14.6.3 Description of ETM registers

An additional description of some of the ETM registers is given in the following sections. See the *ARM Embedded Trace Macrocell Architecture Specification* for more information.

ETM Control Register

The ETM Control Register controls general operation of the ETM, such as whether tracing is enabled.

Reset value: 0x00002411

Implemented bits:

- [21] Port Size[3]
- [17:16] Port Mode[1:0]
- [13] Port Mode[2]

- [11] ETMEN
- [10] ETM Programming
- [9] Debug request control
- [8] Branch Output
- [7] Stall Processor
- [6:4] PortSize [2:0]
- [0] ETM Power Down.

All other bits RAZ, ignore writes.

Configuration Code Register

The ETM Configuration Code Register enables the debugger to read the implementation-specific configuration of the ETM.

Reset value: 0x8C800000

Bits [22:20] are fixed at 0 and not supplied by the ASIC. Bits [18:17] are supplied by the **MAXEXTIN[1:0]** input bus, and read the lower value of MAXEXTIN and the number 2 (the number of EXTINs). This indicates:

- software accesses supported
- trace start/stop block present
- no CID comparators
- FIFOFULL logic is present
- no external outputs
- 0-2 external inputs (controlled by MAXEXTIN)
- no sequencer
- no counters
- no MMDs
- no data comparators
- no address comparator pairs.

System Configuration Register

The System Configuration Register shows the ETM features supported by the ASIC.

Reset value: 0x00020D09

Bits [11:10] are implemented as normal. Bits [9], [2:0] are fixed as 4'b0001.

TraceEnable Control 1 Register

The TraceEnable Control 1 Register is one of the registers that configures TraceEnable.

Only bit [25], Trace start/stop enable, is implemented as follows:

- 0 - Tracing is unaffected by the trace start/stop logic.
- 1 - Tracing is controlled by the trace on and off addresses configured for the trace start/stop logic.

The trace start/stop resource (resource 0x5F) is unaffected by the value of this bit.

ETM ID Register

The ETM ID Register holds the ETM architecture variant, and precisely defines the programmer's model for the ETM.

Reset value: 0x4114F242

This indicates:

- ARM implementor
- special branch encoding, affects bits [7:6] of each byte
- 32-bit Thumb instruction supported
- core family is found elsewhere
- ETMv3.4
- implementation revision 2.

Configuration Code Extension Register

The Configuration Code Extension Register holds additional bits for ETM configuration code. It describes the extended external inputs.

Reset value: 0x00018800

This register indicates:

- start/stop block uses embedded *In Circuit Emulator* (ICE) inputs
- four embedded ICE inputs
- no data comparisons supported
- all registers are readable
- no extended external input supported.

Power Down Status Register

The *Power Down Status Register* (PDSR) indicates whether the ETM is powered up or not.

Reset value: 0x00000001

Only bit [0] is implemented. It indicates whether the ETM debug power domain is powered up or not:

- 0 = ETM debug power domain not powered up
- 1 = ETM debug power domain powered up.

Note

If the ETM is not powered up, the ETM registers are not accessible.

14.6.4 ETM Event resources

The trace enable event and trigger event are configured using the same mechanism. For each event, two resources are defined, together with a boolean function of those two resources. Table 14-10 and Table 14-11 on page 14-23 show these.

Table 14-10 Boolean function encoding for events

Encoding	Function
b000	A
b001	NOT(A)
b010	A AND B
b011	NOT(A) AND B
b100	NOT(A) AND NOT (B)
b101	A OR B
b110	NOT (A) OR B
b111	NOT (A) OR NOT (B)

Table 14-11 Resource identification encoding

Resource type, bits [6:4]	Index range, bits [3:0]	Description of resource type
b010	0-3	DWT Comparator inputs (0-3)
b101	15	Trace Start/Stop resource
b110	0-1	ExtIn (0-1)
b110	15	HardWired (always True)

14.6.5 Cross trigger interface

Table 14-12 and Table 14-13 list the recommended *Cross Trigger Interface* (CTI) connections for Cortex-M3 systems.

Table 14-12 Input connections

Trigger bit	Source signal	Source device	Comments
[7]	ETMTRIGOUT	ETM	Recommended if ETM is present.
[6]	ETMTRIGGER[2]	DWT	Recommended.
[5]	ETMTRIGGER[1]	DWT	Recommended.
[4]	ETMTRIGGER[0]	DWT	Recommended.
[3]	ACQCOMP	ETB	See Full.
[2]	FULL	ETB	Recommended if an ETB is present. If a single ETB is shared between multiple cores, only connect to the CTI of one of the cores.
[1]	User Defined	-	-
[0]	DBGACK	Core	Compulsory.

Table 14-13 Trigger output connections

Trigger bit	Source signal	Source device	Comments
[7]	User defined	-	-
[6]	User defined	-	-
[5]	ETMEXTIN[1]	ETM	Compulsory if ETM is present.

Table 14-13 Trigger output connections (continued)

Trigger bit	Source signal	Source device	Comments
[4]	ETMEXTIN[0]	ETM	Compulsory if ETM is present.
[3]	INTISR[y]	NVIC	See Full.
[2]	INTISR[x]	NVIC	Compulsory. Any interrupt can be used.
[1]	User defined	-	-
[0]	EDBGRQ	Core	Compulsory.

There is no connection from **ETMDBGREQ** of the ETM to the CTL. If required, this signal must be ORed with an external debug request input, and trigger bit [0] from the CTL.

Chapter 15

Embedded Trace Macrocell Interface

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

- *About the ETM interface* on page 15-2
- *CPU ETM interface port descriptions* on page 15-3
- *Branch status interface* on page 15-6.

15.1 About the ETM interface

The ETM interface enables simple connection of an ETM to the processor. It provides a channel for instruction trace to the ETM.

15.2 CPU ETM interface port descriptions

The processor has a port that enables the ETM to determine the instruction execution sequence. These port descriptions are described in Table 15-1.

Table 15-1 ETM interface ports

Port name	Direction	Qualified by	Description
ETMIVALID	Output	No qualifier	Instruction in execute is valid. Marks that an opcode has entered the first cycle of execute.
ETMIBRANCH	Output	ETMIVALID	Opcode is a branch target. Marks that current code is the destination of a <i>Program Counter</i> (PC) modifying event (branch, interrupt processing).
ETMIINDBR	Output	ETMIBRANCH	Opcode branch target is indirect. Marks that the current opcode is a branch target whose destination the PC contents cannot deduce. For example, LSU, register move, or interrupt processing.
ETMDVALID	Output	No qualifier	Signals that the current data address as seen by the <i>Data Watchpoint and Trace</i> (DWT) is valid on this cycle.
ETMICCFAIL	Output	ETMIVALID	Opcode condition code fail or pass. Marks if the current opcode has failed or passed its conditional execution check. An opcode is conditionally executed if it is a conditional branch, or for all other opcode found in an IT block.
ETMINTSTAT[2:0]	Output	No qualifier	Interrupt status. Marks the interrupt status of the current cycle: 000 no status 001 interrupt entry 010 interrupt exit 011 interrupt return 100 - Vector fetch and stack push. ETMINTSTAT Entry/Return is asserted in the first cycle of the new interrupt context. Exit occurs without ETMIVALID .
ETMINTNUM[8:0]	Output	ETMINTSTAT	Interrupt number. Marks the interrupt number of the current execution context.

Table 15-1 ETM interface ports (continued)

Port name	Direction	Qualified by	Description
ETMIA[31:1]	Output	No qualifier	Instruction address. Indicates the current fetch address of the opcode in execution, or of the last opcode executed. You can determine the context by examining: ETMIVALID HALTED SLEEPING . The ETM examines this net when ETMIVALID is asserted. The DWT examines this net for PC samples and bus watching.
ETMFOLD	Output	ETMIVALID	Opcode fold. Indicates that an IT opcode has been folded in this cycle. PC advances past the current (16-bit) opcode and the IT instruction (16 bits). This affects the ETMIA.
ETMFLUSH	Output	No qualifier	Flush marker of PC event. A PC modifying opcode has executed or an interrupt push/pop has started. The ETM can use this control to complete outstanding packets in preparation for an ETMIBRANCH event.
ETMFINDBR	Output	ETMFLUSH	Flush is indirect. Marks that the PC cannot deduce the flush hint destination.
ETMCANCEL	Output	No qualifier	Current opcode in execute has been cancelled. Opcodes that are interrupted restart or continue on return to this execution context. These include: LDR/STR LDRD/STRD LDM/STM U/SMULL MLA U/SDIV MSR CPSID

Table 15-1 ETM interface ports (continued)

Port name	Direction	Qualified by	Description
ETMISTALL	Output	No qualifier	Indicates that the last instruction signalled by the core has not yet entered execute. If ETMICANCEL is asserted with ETMISTALL , it indicates that the stalled instruction did not execute, and the previous instruction was cancelled.
ETMTRIGGER[3:0]	Output	No qualifier	Output trigger from DWT. One bit for each of the four DWT comparators.
ETMTRIGINOTD[3:0]	Output	No qualifier	Output indicates if the ETM is triggered on an instruction or data match.

15.3 Branch status interface

The processor speculatively fetches some branch targets at decode time rather than waiting for the branch to be executed. Because the processor has an internal pre-fetch buffer, incorrect speculative fetches might not incur a penalty depending on the instructions being executed and the number of wait-states on the memory. This means that branches for low wait state memories can be executed in one cycle. Certain implementations might find it beneficial to employ additional pre-fetching logic external to the processor to improve instruction throughput. This logic might be able to use **BRCHSTAT** to improve efficiency. The branch status signal, **BRCHSTAT**, gives information about the opcode currently in decode. **BRCHSTAT** indicates whether the current instruction in decode causes a non-sequential fetch to occur at decode time or execute time, whether the branch is conditional and whether the branch is forward or backwards. Execute time branches might have multicycle **BRCHSTAT**, which is dependent on the stall of the preceding opcode in execute. Table 15-2 describes the signal function.

Table 15-2 Branch status signal function

Name	Direction	Description
BRCHSTAT	Output	0000 = No branch currently in decode
		0001 = Decode time conditional branch backwards currently in decode
		0010 = Decode time conditional branch currently in decode
		0011 = Execute time conditional branch currently in decode
		0100 = Decode time unconditional branch currently in decode
		0101 = Execute time unconditional branch in decode
		0110 = Reserved
		0111 = Reserved
		1000 = Conditional branch in decode taken, cycle after b0001 or b0010

Table 15-3 on page 15-7 shows the branches that the processor can execute. For each type of branch the stage in which the branch is evaluated is shown. For example, all branches with immediates are evaluated during decode. This means that when ever a branch immediate enters the decode stage the branch target address is issued on the AHB.

The ALU register based branches and LSU PC modifying instructions are recognized as conditional branches, b0011, if they are present in IT blocks. Otherwise they are recognized as unconditional branches, b0101.

Table 15-3 Branches and stages evaluated by the processor

Branch Instruction	Instruction size	Stage branch target is issued	Notes
B <imm>	16 bits	Decode	-
B <imm>	32 bits	Decode	-
BL	32 bits	Decode	If LR is not being written during decode.
BLX LR	16 bits	Decode	If LR is not being written during decode.
BX LR	16 bits	Decode	If LR is not being written during decode.
MOV PC, LR	16 bits	Decode	If LR is not being written during decode.
ADD PC	32 bits	Execute	-
BLX	16 bits	Execute	If LR is not the source register or if LR is being written during decode.
BX	16 bits	Execute	If LR is not the source register or if LR is being written during decode.
CBZ, CBNZ	16 bits	Execute	-
ISB	16 bits	Execute	-
LDR PC	32 bits	Execute	-
LDM to PC	32 bits	Execute	-
MOV PC	32 bits	Execute	If LR is not the source register or if LR is being written during decode and LR is the source register.

Table 15-3 Branches and stages evaluated by the processor (continued)

Branch Instruction	Instruction size	Stage branch target is issued	Notes
POP {PC}	16 bits	Execute	-
POP {PC}	32 bits	Execute	-
TBB/TBH	32 bits	Execute	-

————— **Note** —————

- The encoding b1000 is only asserted in the cycle after conditional decode branches if the branch is taken. This is a registered output, so could be used to drive a multiplexor of addresses in the memory controller.
- Multicycle LSU in the b0101 encoding suppresses fetches during execute because it is known that the unconditional branch is executed so sequential fetches are prevented.
- Encodings are present for the multicycle duration of the decode, not only when decode enable is asserted.

Speculative fetches might be cancelled during wait states. This means that the fetch address might change to a new address while **HREADY** is low. See *AMBA 3 compliance* on page 12-3.

Figure 15-1 and Figure 15-2 on page 15-9 show a conditional branch backwards not taken and taken. The branch occurs speculatively in the decode phase of the opcode. The branch target is a halfword unaligned 16-bit opcode.

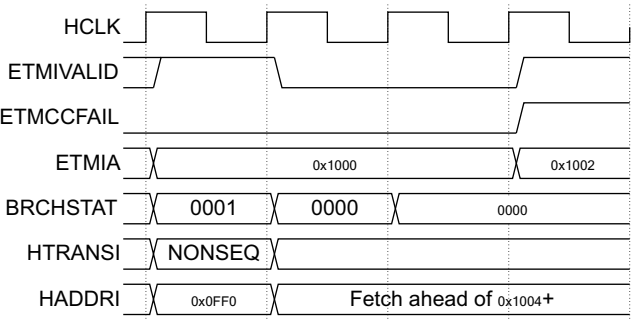


Figure 15-1 Conditional branch backwards not taken

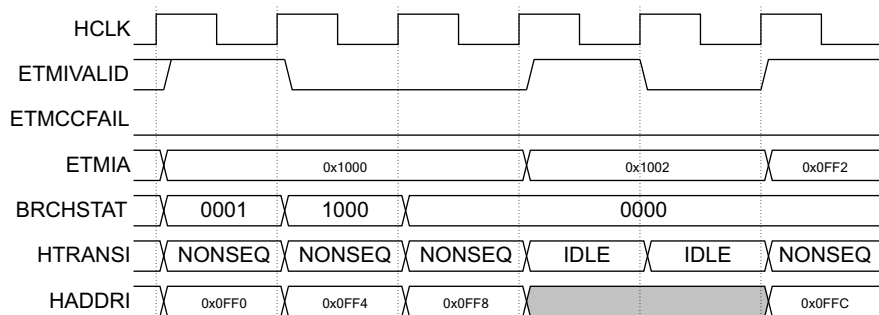


Figure 15-2 Conditional branch backwards taken

Figure 15-3 and Figure 15-4 show a conditional branch forwards not taken and taken. The branch occurs speculatively in the decode phase of the opcode. The branch target is a halfword aligned 16-bit opcode.

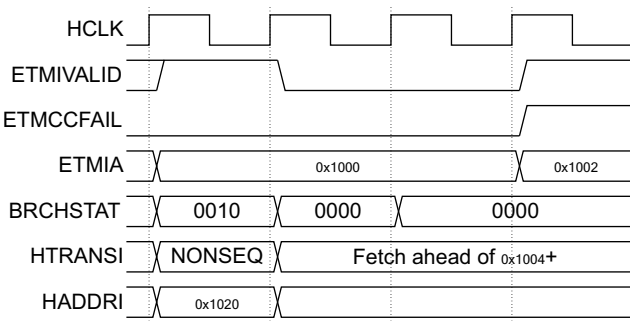


Figure 15-3 Conditional branch forwards not taken

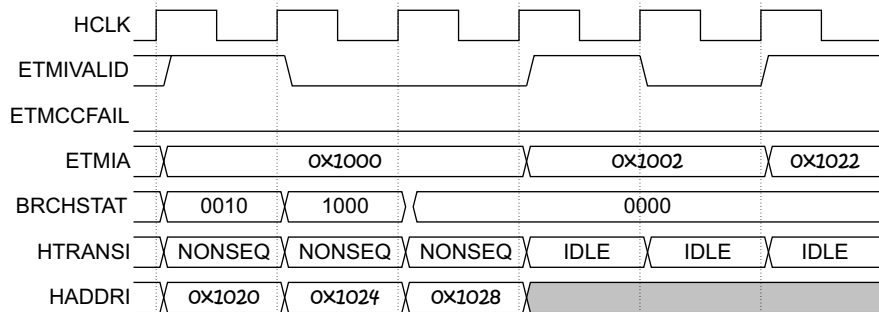


Figure 15-4 Conditional branch forwards taken

Figure 15-5 and Figure 15-6 show an unconditional branch in this cycle, during the execute phase of the preceding opcode without and with pipeline stalls. The branch occurs in the decode phase of the opcode. The branch target is an aligned 32-bit opcode.

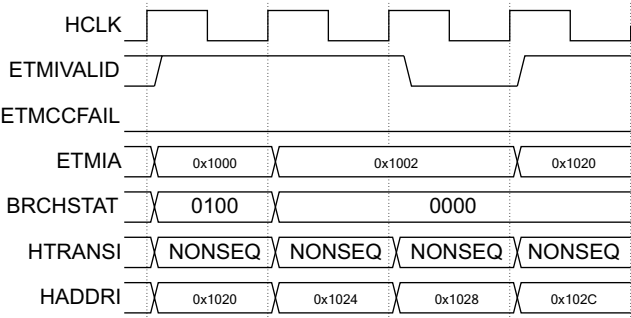


Figure 15-5 Unconditional branch without pipeline stalls

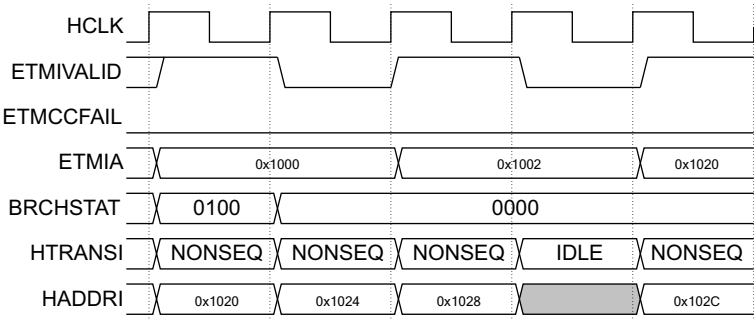


Figure 15-6 Unconditional branch with pipeline stalls

Figure 15-7 on page 15-11 and Figure 15-8 on page 15-11 show an unconditional branch in the next opcode. The branch occurs in the execute phase of the opcode. The branch target is an aligned and unaligned 32-bit ALU opcode.

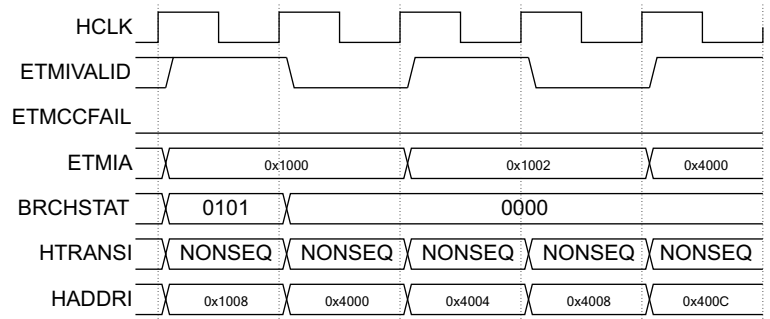
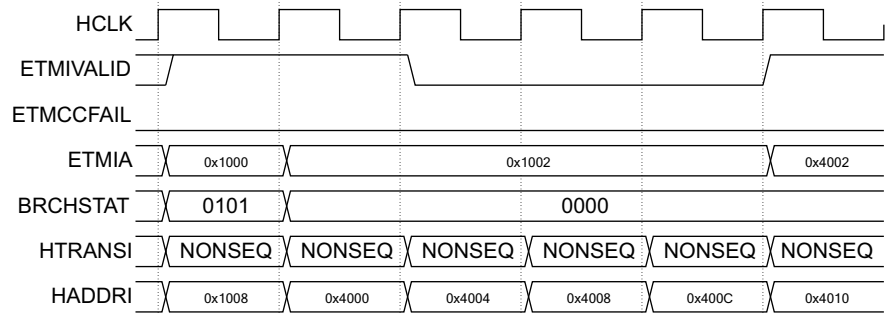
**Figure 15-7 Unconditional branch in execute aligned****Figure 15-8 Unconditional branch in execute unaligned**

Table 15-4 shows an example of an opcode sequence.

Table 15-4 Example of an opcode sequence

Execute cycle	Fetch address	Opcode
1	0x1020	ADD r1,#1
2	0x1022	LDR r3,[r4]
3	0x1024	ADD r2,#3
4	0x1026	CMP r3,r2
5	0x1028	BEQ = Target1
6	0x1040	CMP r1,r2
7	0x1042	ITE EQ // folded
8	0x1044	LDR EQ r3,[r4,r1] // skipped

Table 15-4 Example of an opcode sequence (continued)

Execute cycle	Fetch address	Opcode
9	0x1046	LDR NE r3,[r4,r2]
10	0x1048	ADD r6,r3
11	0x104A	NOP
12	0x104C	BX r14
13	0x0FC4	CMP
14	0x0FC6	BEQ = Target2 // not taken
15	0x0FC8	BX r5

Figure 15-9 on page 15-13 shows the timing sequence for the example opcode sequence in Table 15-4 on page 15-11.

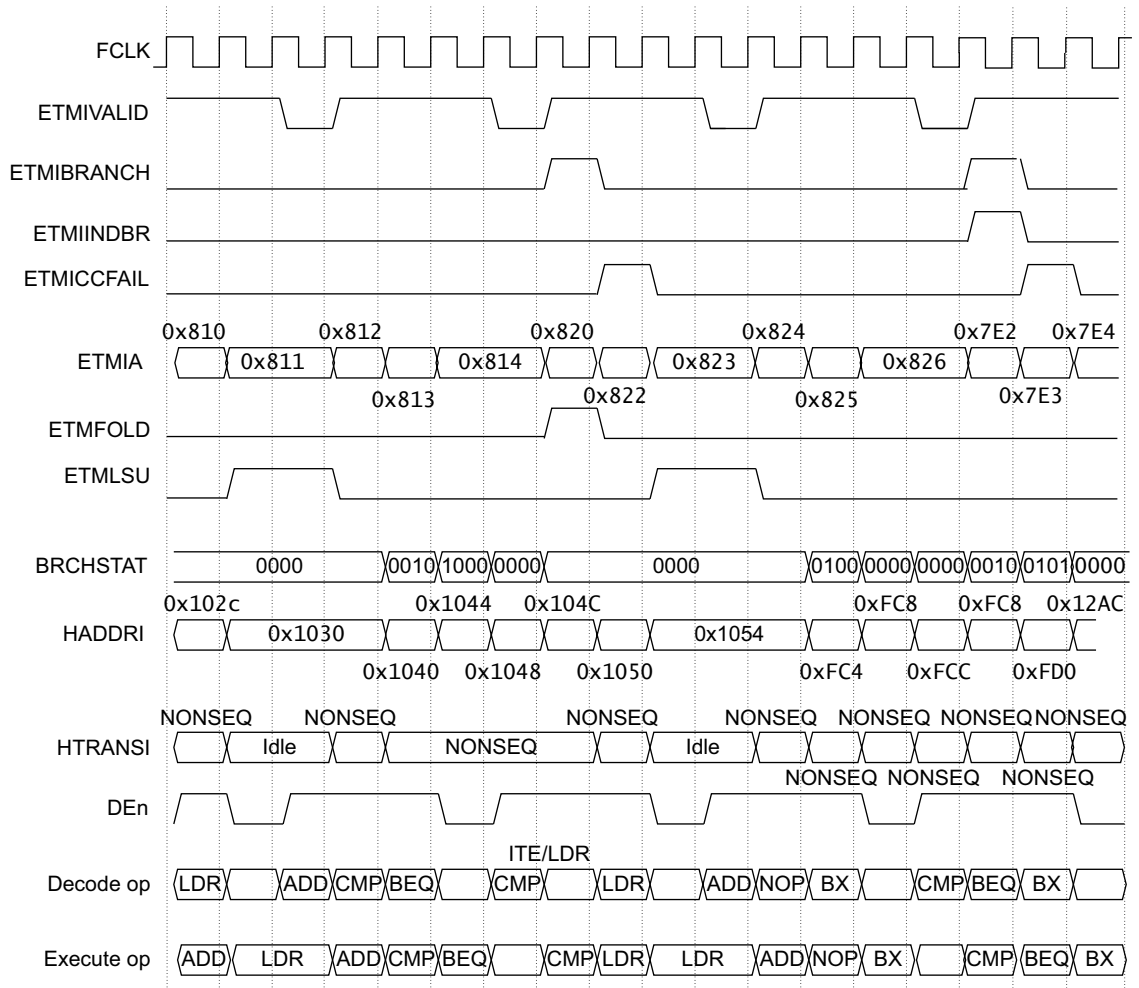


Figure 15-9 Example of an opcode sequence

Chapter 16

AHB Trace Macrocell Interface

This chapter describes the *Advanced High-performance Bus* (AHB) trace macrocell interface. It contains the following sections:

- *About the AHB trace macrocell interface* on page 16-2
- *CPU AHB trace macrocell interface port descriptions* on page 16-3.

16.1 About the AHB trace macrocell interface

The *AHB Trace Macrocell* (HTM) interface enables a simple connection of the AHB trace macrocell to the processor. It provides a channel for the data trace to the HTM.

To use the HTM interface, the trace level must be set to level 3 before implementation. TRCENA must also be set to 1 before you enable the HTM to enable the HTM port to supply trace data.

16.2 CPU AHB trace macrocell interface port descriptions

Table 16-1 list the AHB interface ports.

Table 16-1 AHB interface ports

Port name	Direction	Description
HTMDHADDR[31:0]	Output	32-bit address.
HTMDHTRANS[1:0]	Output	Output indicates the type of the current data transfer. Can be IDLE, NONSEQUENTIAL, OR SEQUENTIAL.
HTMDHSIZE[1:0]	Output	Indicates the size of the access. Can be 8, 16, or 32 bits.
HTMDHBURST[2:0]	Output	Output indicates if the transfer is part of a burst.
HTMDHPROT[3:0]	Output	Provides information on the access.
HTMDHWDATA[31:0]	Output	32-bit write data bus.
HTMDHWRITE	Output	Write not read.
HTMDHRDATA[31:0]	Output	Read data bus.
HTMDHREADY	Output	When HIGH indicates that a transfer has completed on the bus. The signal is driven LOW to extend a transfer.
HTMDHRESP[1:0]	Output	The transfer response status. OKAY or ERROR.
HTMDHADDR[31:0]	Output	32-bit address.
HTMDHTRANS[1:0]	Output	Output indicates the type of the current data transfer. Can be IDLE, NONSEQUENTIAL, OR SEQUENTIAL.

Chapter 17

Trace Port Interface Unit

This chapter describes the *Trace Port Interface Unit* (TPIU). It contains the following sections:

- *About the TPIU* on page 17-2
- *TPIU registers* on page 17-8
- *Serial wire output connection* on page 17-21.

17.1 About the TPIU

The TPIU is an optional component that acts as a bridge between the on-chip trace data from the *Embedded Trace Macrocell* (ETM) and the *Instrumentation Trace Macrocell* (ITM), with separate IDs, to a data stream, encapsulating IDs where required, that is then captured by a *Trace Port Analyzer* (TPA).

The TPIU is specially designed for low-cost debug. It is a special version of the CoreSight TPIU, and you can replace it with CoreSight components if system requirements demand the additional features of the CoreSight TPIU.

There are two configurations of the TPIU:

- A configuration that supports ITM debug trace.
- A configuration that supports both ITM and ETM debug trace.

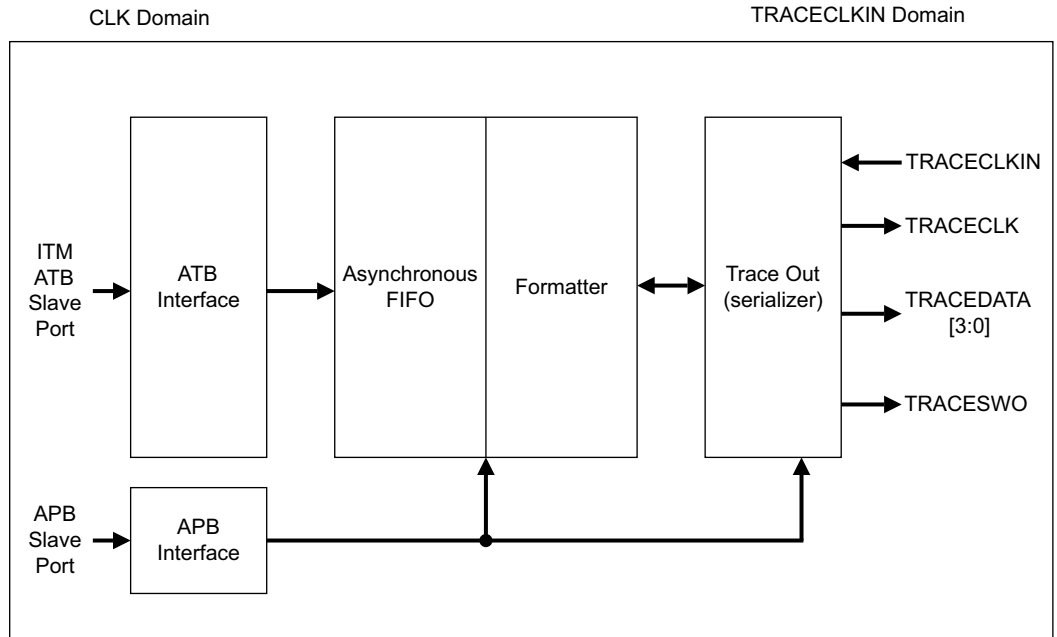
If the implementation requires no trace support then the TPIU might not be present.

Note

If your Cortex-M3 system uses the optional ETM component, you must use the TPIU configuration that supports both ITM and ETM debug trace. For a full description of the ETM, see Chapter 14 *Embedded Trace Macrocell*.

17.1.1 TPIU block diagrams

Figure 17-1 on page 17-3 and Figure 17-2 on page 17-4 show the component layout of the TPIU for both configurations.

**Figure 17-1 TPIU block diagram (non-ETM version)**

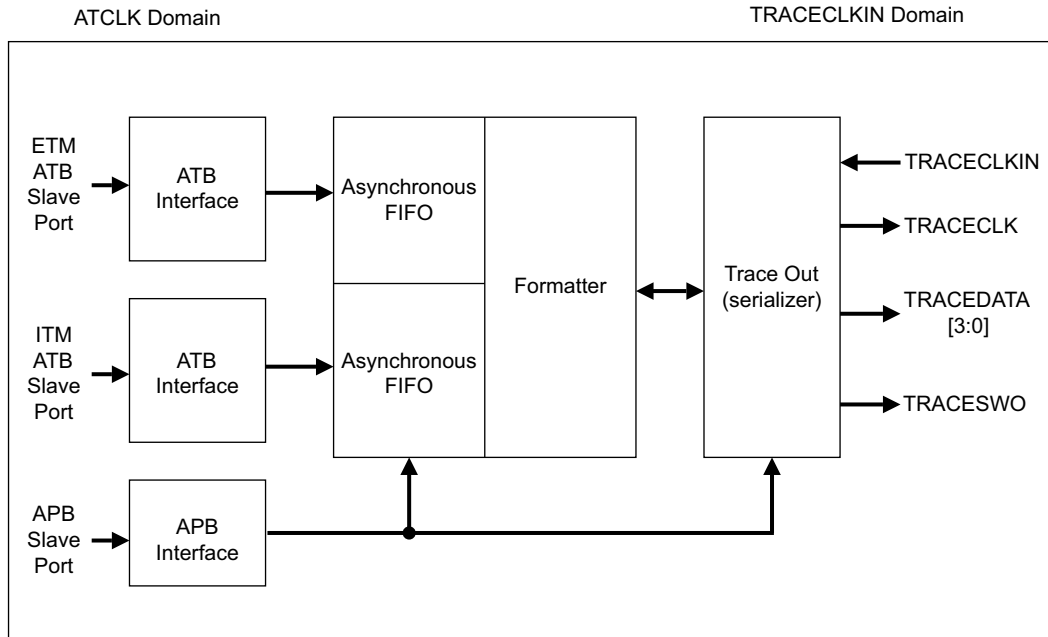


Figure 17-2 TPIU block diagram (ETM version)

17.1.2 TPIU components

A description of the main components of the TPIU is given in the following sections:

- *Asynchronous FIFO*
- *Formatter*
- *Trace out* on page 17-5
- *Advanced Trace Bus interface* on page 17-5
- *Advanced Peripheral Bus interface* on page 17-5.

Asynchronous FIFO

The asynchronous FIFO enables trace data to be driven out at a speed that is not dependent on the speed of the core clock.

Formatter

The formatter inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source. The formatter is always active when the TRACEPORT mode is active.

Trace out

The trace out block serializes formatted data before it goes off-chip.

Advanced Trace Bus interface

The TPIU accepts trace data from a trace source, either direct from a trace source (ETM or ITM) or using a Trace Funnel. For more information, see *Advanced Trace Bus interface*.

Advanced Peripheral Bus interface

The APB interface is the programming interface for the TPIU. For more information, see *Advanced Peripheral Bus interface*.

17.1.3 TPIU inputs and outputs

This section describes the TPIU inputs and outputs. It contains the following:

- *Trace out port*
- *Advanced Trace Bus interface* on page 17-6.
- *Miscellaneous configuration inputs* on page 17-6
- *APB interface* on page 17-7.

Trace out port

Table 17-1 describes the trace out port signals.

Table 17-1 Trace out port signals

Name	Type	Description
TRACECLKIN	Input	Decoupled clock from ATB to enable easy control of the trace port speed. Typically this is derived from a controllable clock source on chip, but an external clock generator could drive it if a high speed pin is used. Data changes on the rising edge only.
TRESETn	Input	This is a reset signal for the TRACECLKIN domain. This signal is typically driven from Power on Reset, and must be synchronized to TRACECLKIN .
TRACECLK	Output	TRACEDATA changes on both edges of TRACECLK .
TRACEDATA[3:0]	Output	Output data for clocked modes.
TRACESWO	Output	Output data for asynchronous modes.

Advanced Trace Bus interface

There is one or two ATB interfaces depending on the TPIU configuration. Table 17-2 describes the ATB port signals. The signals for port 2 are not used when the TPIU is configured with a single ATB interface.

Table 17-2 ATB port signals

Name	Type	Description
CLK	Input	Trace bus and APB interface clock.
nRESET	Input	Reset for the CLK domain (ATB/APB interface).
CLKEN	Input	Clock enable for CLK domain.
ATVALID1S	Input	Data from trace source 1 is valid in this cycle.
ATREADY1S	Output	If this signal is asserted (ATVALID high), then the data was accepted this cycle from trace source 1.
ATDATA1S[7:0]	Input	Trace data input from source 1.
ATID1S[6:0]	Input	Trace source ID for source 1. This must not change dynamically.
ATVALID2S	Input	Data from trace source 2 is valid in this cycle.
ATREADY2	Output	If this signal is asserted (ATVALID high), then the data was accepted this cycle from trace source 2.
ATDATA2S[7:0]	Input	Trace data input from source 2.
ATID2S[6:0]	Input	Trace source ID for source 2. This must not change dynamically.

Miscellaneous configuration inputs

Table 17-3 describes the miscellaneous configuration inputs.

Table 17-3 Miscellaneous configuration inputs

Name	Type	Description
MAXPORTSIZE[1:0]	Input	Defines the maximum number of pins available for synchronous trace output.
SyncReq	Input	Global trace synchronization trigger. Inserts synchronization packets into the formatted data stream. Only used when the formatter is active. This signal must be connected to the DSYNC output from Cortex-M3.
TRIGGER	Input	Causes a trigger packet to be inserted into the trace stream when the formatter is active.

Table 17-3 Miscellaneous configuration inputs (continued)

Name	Type	Description
SWOACTIVE	Output	SWO mode selected. Use for pin multiplexing.
TPIUACTIV	Output	Indicates that the TPIU has data that is in the process of being output.
TPIUBAUD	Output	Toggles at baud frequency (in TRACECLKIN domain).

APB interface

Table 17-4 describes the APB interface inputs.

Table 17-4 APB interface

Name	Type	Description
PSEL	Input	Peripheral select
PWRITE	Input	Peripheral write control
PENABLE	Input	Peripheral transfer enable
PADDR[11:2]	Input	Peripheral address
PWDATA[31:0]	-	Write data
PRDATA[31:0]	-	Read data

17.2 TPIU registers

This section describes the TPIU registers. It contains the following:

- *Summary of the TPIU registers*
- *Description of the TPIU registers on page 17-9.*

17.2.1 Summary of the TPIU registers

Table 17-5 provides a summary of the TPIU registers.

———— Note —————

You can configure any of the TPIU registers to be present or not present. Any register that is configured as not present reads as zero.

Table 17-5 TPIU registers

Name of register	Type	Address	Reset value	Page
Supported Sync Port Sizes Register	Read-only	0xE0040000	0bxx0x	page 17-9
Current Sync Port Size Register	Read/write	0xE0040004	0x01	page 17-10
Async Clock Prescaler Register	Read/write	0xE0040010	0x0000	page 17-10
Selected Pin Protocol Register	Read/write	0xE00400F0	0x01	page 17-11
Formatter and Flush Status Register	Read-only	0xE0040300	0x08	page 17-11
Formatter and Flush Control Register	Read/write	0xE0040304	0x102	page 17-12
Formatter Synchronization Counter Register	Read-only	0xE0040308	0x00	page 17-14
Integration Register: TRIGGER	Read-only	0xE0040EE8	0x0	page 17-17
Integration Register: ITATBCTR2	Read-only	0xE0040EF0	0x0	page 17-15
Integration Register: ITATBCTR0	Read-only	0xE0040EF8	0x0	page 17-15
Integration Mode Control Register	Read/write	0xE0040F00	0x0	page 17-16
Integration register : FIFO data 0	Read only	0xE0040EEC	0x--000000	page 17-17
Integration register : FIFO data 1	Read only	0xE0040EFC	0x--000000	page 17-18
Claim tag set register	Read/write	0xE0040FA0	0xF	page 17-20
Claim tag clear register	Read/write	0xE0040FA4	0x0	page 17-19

Table 17-5 TPIU registers (continued)

Name of register	Type	Address	Reset value	Page
Device ID register	Read only	0xE0040FC8	0xCA0 (ETM present) 0xCA1 (ETM not present)	page 17-20
PID4	Read only	0xE0040FD0	0x04	-
PID5	Read only	0xE0040FD4	0x00	-
PID6	Read only	0xE0040FD8	0x00	-
PID7	Read only	0xE0040FDC	0x00	-
PID0	Read only	0xE0040FE0	0x23	-
PID1	Read only	0xE0040FE4	0xB9	-
PID2	Read only	0xE0040FE8	0x2B	-
PID3	Read only	0xE0040FEC	0x00	-
CID0	Read only	0xE0040FF0	0x0D	-
CID1	Read only	0xE0040FF4	0x90	-
CID2	Read only	0xE0040FF8	0x05	-
CID3	Read only	0xE0040FFC	0xB1	-

17.2.2 Description of the TPIU registers

This section describes the TPIU registers.

Supported Sync Port Sizes Register

This register is read/write. Each bit location represents a single port size that is supported on the device, that is, 4, 2 or 1 in bit locations [3:0]. If the bit is set then that port size is permitted. By default the RTL is designed to support all port sizes, set to 0x0000000B. This register is constrained by the input tie-off **MAXPORTSIZE**. The external tie-off, **MAXPORTSIZE**, must be set during finalization of the ASIC to reflect the actual number of **TRACEDATA** signals wired to physical pins. This is to ensure that tools do not attempt to select a port width that an attached TPA cannot capture. The value on **MAXPORTSIZE** causes bits within the Supported Port Size register that represent wider widths to be clear, that is, unsupported.

Figure 17-3 shows the bit assignments.

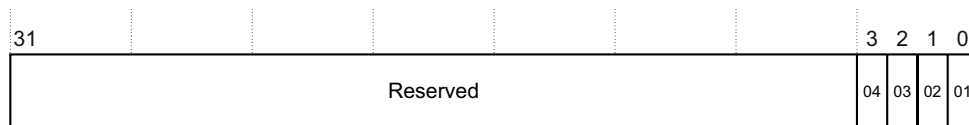


Figure 17-3 Supported Sync Port Size Register bit assignments

Current Sync Port Size Register

This register is read/write. The Current Sync Port Size Register has the same format as the Supported Sync Port Sizes Register but only one bit is set, and all others must be zero. Writing values with more than one bit set, or setting a bit that is not indicated as supported is not supported and causes Unpredictable behavior.

It is more convenient to use the same format as the Supported Sync Port Sizes Register because it saves on having to decode the sizes later on in the device, and also maintains the format from the other register bank for checking for valid assignments.

On reset this defaults to the smallest possible port size, 1 bit, and so reads as 0x00000001.

Async Clock Prescaler Register

Use the Async Clock Prescaler Register to scale the baud rate of the asynchronous output.

Figure 17-4 shows the bit assignments of the Async Clock Prescaler Register.

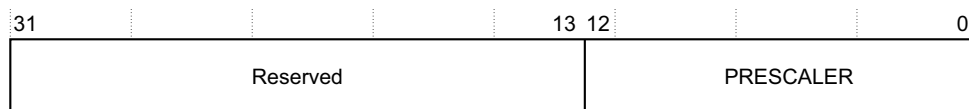


Figure 17-4 Async Clock Prescaler Register bit assignments

Table 17-6 describes the bit assignments of the Async Clock Prescaler Register.

Table 17-6 Async Clock Prescaler Register bit assignments

Bits	Field	Function
[31:13]	-	Reserved. RAZ/SBZP.
[12:0]	PRESCALER	Divisor for TRACECLKIN is Prescaler + 1.

Selected Pin Protocol Register

Use the Selected Pin Protocol Register to select which protocol to use for trace output.

The register address, access type, and Reset state are:

Address 0xE00400F0

Access Read/write

Reset state 0x01

Figure 17-5 shows the bit assignments of the Selected Pin Protocol Register.

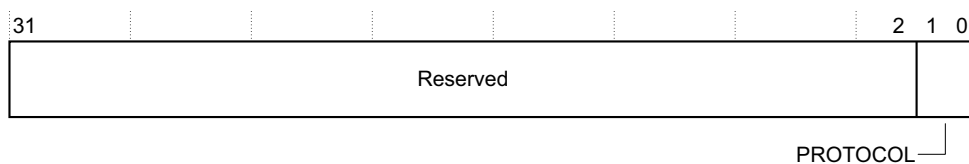


Figure 17-5 Selected Pin Protocol Register bit assignments

Table 17-7 describes the bit assignments of the Selected Pin Protocol Register.

Table 17-7 Selected Pin Protocol Register bit assignments

Bits	Field	Function
[31:2]	-	Reserved
[1:0]	PROTOCOL	00 - TracePort mode 01 - SerialWire Output (Manchester). This is the reset value. 10 - SerialWire Output (NRZ) 11 - Reserved.

Note

If this register is changed while trace data is being output, data corruption occurs.

Formatter and Flush Status Register

Use the Formatter and Flush Status Register to read the status of TPIU formatter.

The register address, access type, and Reset state are:

Address 0xE0040300

Access Read only

Reset state 0x08

Figure 17-6 shows the bit assignments of the Formatter and Flush Status Register.

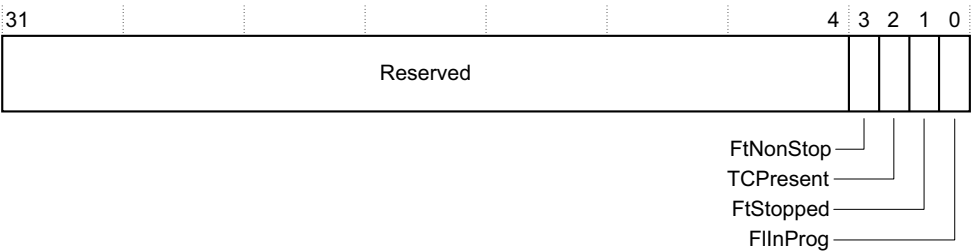


Figure 17-6 Formatter and Flush Status Register bit assignments

Table 17-8 describes the bit assignments of the Formatter and Flush Status Register.

Table 17-8 Formatter and Flush Status Register bit assignments

Bits	Field	Function
[31:4]	-	Reserved
[3]	FtNonStop	Formatter cannot be stopped
[2]	TCPresent	This bit always reads zero
[1]	FtStopped	This bit always reads zero
[0]	FIInProg	This bit always reads zero

Formatter and Flush Control Register

The Formatter and Flush Control Register.

The register address, access type, and Reset state are:

Address 0xE0040304
Access Read/write
Reset state 0x102

Figure 17-7 on page 17-13 shows the bit assignments of the Formatter and Flush Control Register.

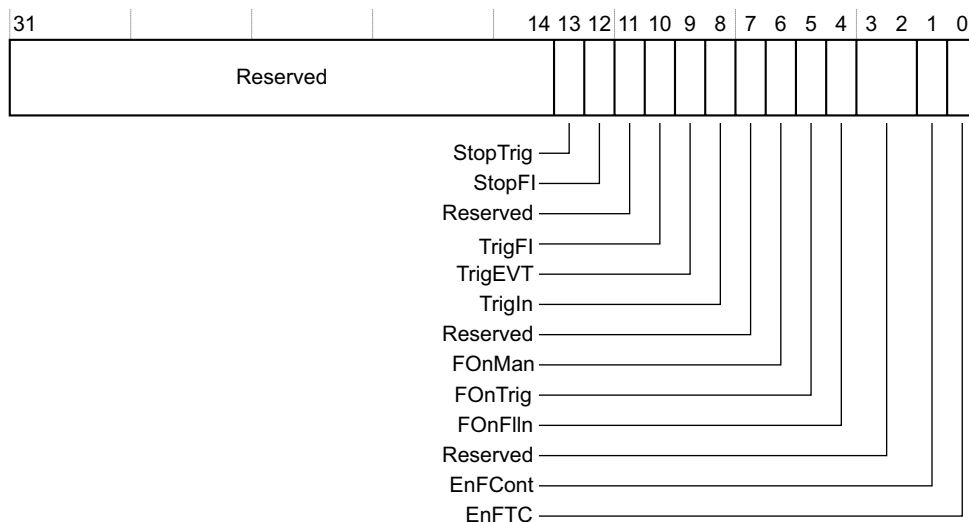


Figure 17-7 Formatter and Flush Control Register bit assignments

Table 17-9 describes the bit assignments of the Formatter and Flush Control Register.

Table 17-9 Formatter and Flush Control Register bit assignments

Bits	Field	Function
[31:14]	-	Reserved.
[13]	StopTrig	Stop the formatter after a Trigger Event is observed.
[12]	StopFI	Stop the formatter after a flush completes.
[11]	-	Reserved.
[10]	TrigFI	Indicates a trigger on Flush completion.
[9]	TrigEVT	Indicate a trigger on a Trigger Event.
[8]	TrigIN	Indicate a trigger on TRIGIN being asserted.
[7]	-	Reserved.
[6]	FOnMan	Manually generate a flush of the system.
[5]	FOnTrig	Generate flush using Trigger event.
[4]	FOnFlIn	Generate flush using the FLUSHIN interface.

Table 17-9 Formatter and Flush Control Register bit assignments (continued)

Bits	Field	Function
[3:2]	-	Reserved.
[1]	EnFCont	Continuous Formatting, no TRACECTL . This bit is set on reset.
[0]	EnFTC	Enable Formatting. Because TRACECTL is never present, this bit reads as zero.

Bit [8] of this register is always set to indicate that triggers are indicated when **TRIGGER** is asserted.

When one of the two single wire output modes is selected, bit [1] of this register enables the formatter to be bypassed. If the formatter is bypassed, only the ITM/DWT trace source (ATDATA2) passes through. The TPIU accepts and discards data that is presented on the ETM port (ATDATA1). This function is intended to be used when it is necessary to connect a device containing an ETM to a trace capture device that is only able to capture Serial Wire Output data. Enabling or disabling the formatter causes momentary data corruption.

———— **Note** ————

If the selected pin protocol register is set to 0x00 (TracePort mode), the Formatter and Flush Control Register always reads 0x102, because the formatter is automatically enabled. If one of the serial wire modes is then selected, the register reverts to its previously programmed value.

Formatter Synchronization Counter Register

The global synchronization trigger is generated by the *Program Counter* (PC) Sampler block. This means that there is no synchronization counter in the TPIU.

The register address, access type, and Reset state are:

Address 0xE0040308
Access Read only
Reset state 0x00

Integration Test Registers

Use the Integration Test Registers to perform topology detection of the TPIU with other devices in a Cortex-M3 system. These registers enable direct control of outputs and the ability to read the value of inputs. The processor provides two Integration Test Registers:

- Integration Test Register - ITATBCTR2
- Integration Test Register - ITATBCTR0.

Integration Test Register-ITATBCTR2

The register address, access type, and Reset state are:

Address 0xE0040EF0

Access Read only

Reset state 0x0

Figure 17-8 shows the bit assignments of the Integration Test Register bit assignments.

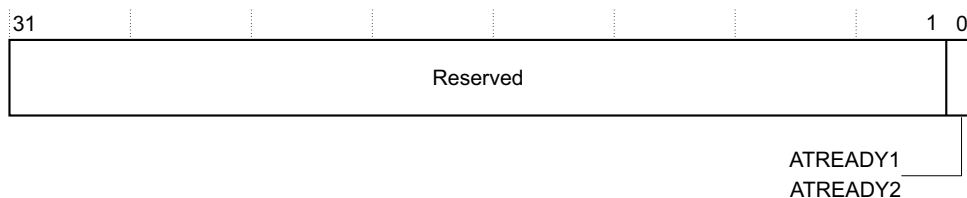


Figure 17-8 Integration Test Register-ITATBCTR2 bit assignments

Table 17-10 describes the bit assignments of the Integration Test Register bit assignments.

Table 17-10 Integration Test Register-ITATBCTR2 bit assignments

Bits	Field	Function
[31:1]	-	Reserved.
[0]	ATREADY1, ATREADY2	This bit reads or sets the value of ATREADY1 and ATREADY2.

Integration Test Register-ITATBCTR0

The register address, access type, and Reset state are:

Address 0xE0040EF8

Access Read only

Reset state 0x0

Figure 17-9 shows the bit assignments of the Integration Test Register bit assignments.

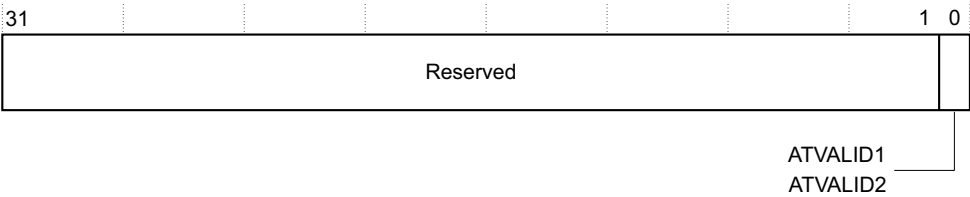


Figure 17-9 Integration Test Register-ITATBCTR0 bit assignments

Table 17-11 describes the bit assignments of the Integration Test Register bit assignments.

Table 17-11 Integration Test Register-ITATBCTR0 bit assignments

Bits	Field	Function
[31:1]	-	Reserved
[0]	ATVALID1, ATVALID2	This bit reads or sets the value of ATVALIDS1 OR-ed with ATVALIDS2.

Integration Mode Control Register

The Integration Mode Control Register enables topology detection.

The register address, access type, and Reset state are:

Address 0xE0040F00

Access Read/write

Reset state 0x0

Figure 17-10 shows the bit assignments of the Integration Mode Control Register.

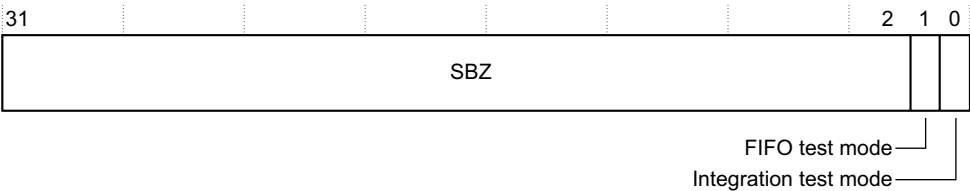


Figure 17-10 Integration Mode Control Register bit assignments

Table 17-12 lists the bit assignments of the Integration Mode Control Register

Table 17-12 Integration Mode Control Register bit assignments

Bits	Field	Function
[31:2]	-	Reserved, SBZ
[1]	FIFO test mode	Enables FIFO test mode
[0]	Integration test mode	Enables integration test mode

Integration Register: TRIGGER

The register address, access type, and Reset state are:

Address 0xE0040EE8

Access Read only

Reset state 0x0

Figure 17-11 shows the bit assignments of the Integration Register : TRIGGER.

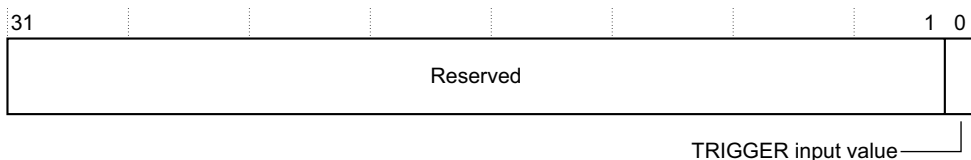


Figure 17-11 Integration Register : TRIGGER bit assignments

Table 17-13 lists the bit assignments of the Integration Register : TRIGGER bit assignments.

Table 17-13 Integration Register : TRIGGER bit assignments

Bits	Field	Function
[31:1]	-	Reserved
[0]	TRIGGER input value	Enables the TRIGGER input

Integration Register : FIFO data 0

The register address, access type, and Reset state are:

Address 0xE0040EEC

Access Read only

Reset state 0x0

Figure 17-12 shows the bit assignments of the Integration register : FIFO data 0.

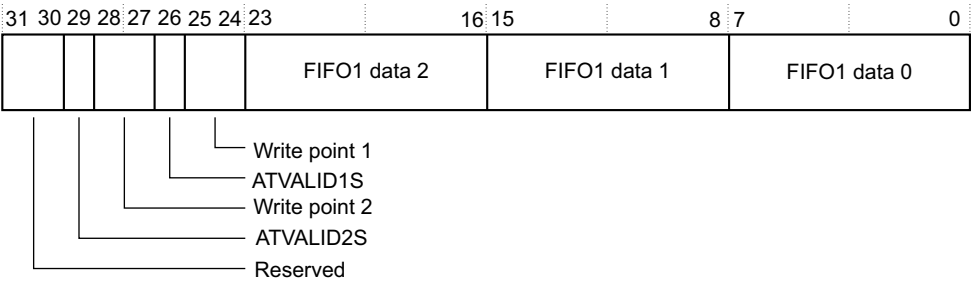


Figure 17-12 Integration register : FIFO data 0 bit assignments

Table 17-14 lists the bit assignments of the Integration register : FIFO data 0.

Table 17-14 Integration register : FIFO data 0 bit assignments

Bits	Field	Function
[31:30]	-	Reserved
[29]	ATVALID2S	
[28:27]	Write point 2	
[26]	ATVALID1S	
[25:24]	Write point 1	
[23:16]	FIFO1 data 2	
[15:8]	FIFO1 data 1	
[7:0]	FIFO1 data 0	

Integration Register : FIFO data 1

The register address, access type, and Reset state are:

Address 0xE0040EFC

Access Read only

Reset state 0x0

Figure 17-13 on page 17-19 shows the bit assignments of the Integration register : FIFO data 1.

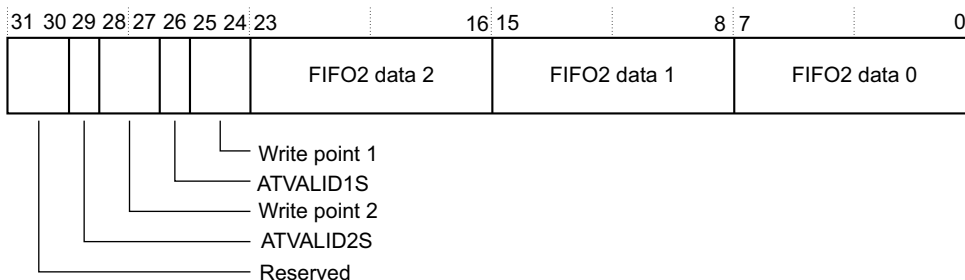
**Figure 17-13 Integration register : FIFO data 1 bit assignments**

Table 17-15 lists the bit assignments of the Integration register : FIFO data 0.

Table 17-15 Integration register : FIFO data 1 bit assignments

Bits	Field	Function
[31:30]	-	Reserved
[29]	ATVALID2S	
[28:27]	Write point 2	
[26]	ATVALID1S	
[25:24]	Write point 1	
[23:16]	FIFO2 data 2	
[15:8]	FIFO2 data 1	
[7:0]	FIFO2 data 0	

CoreSight specific registers

This section describes the CoreSight specific registers.

Claim Tag Clear Register

The register address, access type, and Reset state are:

Address 0xE0040FA4

Access Read/write

Reset state 0x0

This register forms one half of the Claim Tag value. This location enables individual bits to be cleared, write, and returns the current Claim Tag value, read.

The width (n) of this register can be determined from reading the Claim Tag Set Register.

Read Current Claim Tag Value
Write Each bit is considered separately:
 0 = no effect
 1 = clear this bit in the claim tag.

Claim Tag Set Register

The register address, access type, and Reset state are:

Address 0xE0040FA0
Access Read/write
Reset state 0x0

This register forms one half of the Claim Tag value. This location enables individual bits to be set, write, and returns the number of bits that can be set, read.

Read Each bit is considered separately:
 0 = this claim tag bit is not implemented
 1 = this claim tag bit is implemented.
Write Each bit is considered separately:
 0 = no effect
 1 = set this bit in the claim tag.

Device ID Register

The register address, access type, and Reset state are:

Address 0xE0040FC8
Access Read only
Reset state 0xCA0/0xCA1

This register returns:

- 0xCA0 if there is no ETM present.
- 0xCA1 if there is no ETM present.

17.3 Serial wire output connection

The Cortex-M3 TPIU provides a serial wire output mode that requires a single external pin. There are three options available to connect this pin:

- *A dedicated pin can be used for TRACESWO*
- *SWO shared with TRACEPORT*
- *SWO Shared with JTAG-TDO on page 17-22.*

17.3.1 A dedicated pin can be used for TRACESWO

This is the simplest option, but it requires an extra package pin. Figure 17-14 shows the dedicated pin option.

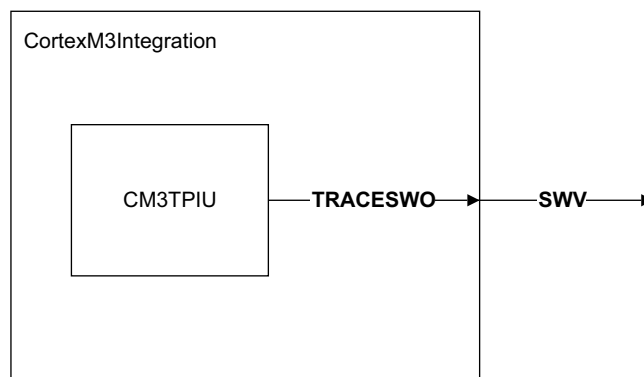


Figure 17-14 Dedicated pin used for TRACESWO

17.3.2 SWO shared with TRACEPORT

A pin can be shared between **TRACEDATA[0]** and **TRACESWO**. Because only one of these two pins can be in use at any one time, there is no loss of functionality using this option, and this is the preferred option when a dedicated trace port is present on the package.

To implement this option, the **SWOACTIVE** output from Cortex-M3 TPIU is used to control the multiplexor. Figure 17-15 on page 17-22 shows the SWO shared with TRACEPORT option.

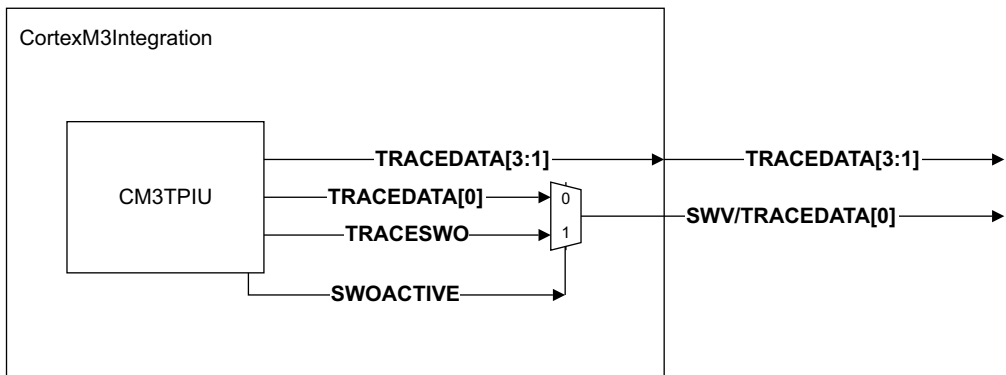


Figure 17-15 SWO shared with TRACEPORT

17.3.3 SWO Shared with JTAG-TDO

For minimal pin count, it is possible to overlay JTAG debug and SWO on the same package pin. This approach is only recommended where there is no provision for a conventional trace port, or for use with more complex system-level debug configuration controls.

If this option is chosen, the Instrumentation Trace is not accessible while the debug port is being used in a JTAG configuration. Serial wire debug and SWO can be used together at the same time.

To implement this option, the **JTAGNSW** output from SWJ-DP is used to control the multiplexor. Figure 17-16 shows the SWO shared with JTAG-TDO option.

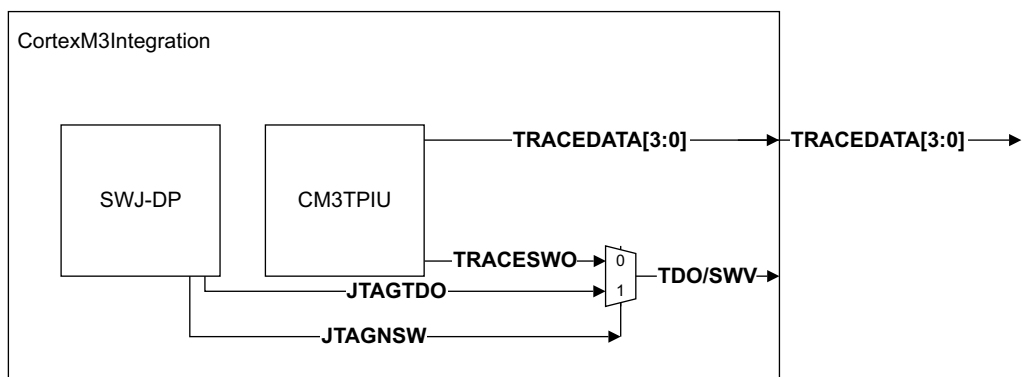


Figure 17-16 SWO shared with JTAG-TDO

Chapter 18

Instruction Timing

This chapter describes the instruction timings of the processor. It contains the following sections:

- *About instruction timing* on page 18-2
- *Processor instruction timings* on page 18-3
- *Load-store timings* on page 18-7.

18.1 About instruction timing

The timing information in this chapter covers each instruction in addition to interactions between instructions. It also contains information about factors that influence timings.

When looking at timings, it is important to understand the role that the system architecture plays. Every instruction must be fetched and every load/store must go out to the system. These factors are described along with intended system design, and the implications for timing.

18.2 Processor instruction timings

Table 18-1 shows the Thumb subset supported in the ARMv7-M architecture. It provides cycle information including annotations to explain how instruction stream interactions affect timing. System effects, such as running code from slower memory, are also considered.

Table 18-1 Instruction timings

Instruction type	Size	Cycles count	Description
Data operations	16	1 (+P ^a if PC is destination)	ADC, ADD, AND, ASR, BIC, CMN, CMP, CPY, EOR, LSL, LSR, MOV, MUL, MVN, NEG, ORR, ROR, SBC, SUB, TST, REV, REV16, REVSH, SXTB, SXTH, UXTB, and UXTH. MUL is one cycle.
Branches	16	1+P ^a	B<cond>, B, BL, BX, and BLX. No BLX with immediate. If branch taken, pipeline reloads (two cycles are added).
Load-store Single	16	2 ^b (+P ^a if PC is destination)	LDR, LDRB, LDRH, LDRSB, LDRSH, STR, STRB, and STRH, and T variants.
Load-store Multiple	16	1+N ^b (+P ^a if PC loaded)	LDMIA, POP, PUSH, and STMIA.
Exception generating	16	-	BKPT stops in debug if debug enabled, fault if debug disabled. SVC faults to SVCcall handler, see <i>ARMv7-M architecture specification</i> for details.
Data operations with immediate	32	1 (+P ^a if PC is destination)	ADC{S}, ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}, and MVN{S}.
Data operations with large immediate	32	1	MOVW, MOVT, ADDW, and SUBW. MOVW and MOVT have a 16-bit immediate (so can replace literal loads from memory). ADDW and SUBW have a 12-bit immediate (so also can replace many from memory literal loads).
Bit-field operations	32	1	BFI, BFC, UBFX, and SBFX. These are bitwise operations that enable control of position and size in bits. These both support C/C++ bit fields (in structs) in addition to many compare and some AND/OR assignment expressions.
Data operations with 3 register	32	1 (+P ^a if PC is destination)	ADC{S}, ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}, and MVN{S}. No PKxxx instructions.

Table 18-1 Instruction timings (continued)

Instruction type	Size	Cycles count	Description
Shift operations	32	1	ASR{S}, LSL{S}, LSR{S}, ROR{S}, and RRX{S}.
Miscellaneous	32	1	REV, REV16, REVSH, RBIT, CLZ, SXTB, SXTH, UXTB, and UXTH. Extension instructions same as corresponding ARM v6 16-bit instructions.
Table Branch	16	4+P ^a	Table branches for switch/case use. These are LDR with shifts and then branch.
Multiply	32	1 or 2	MUL, MLA, and MLS. MUL is one cycle and MLA and MLS are two cycles.
Multiply with 64-bit result	32	3-7 ^c	UMULL, SMULL, UMLAL, and SMLAL. Cycle count based on input sizes. That is, ABS(inputs) < 64K terminates early.
Load-store addressing	32	-	Supports Format PC+/-imm12, Rbase+imm12, Rbase+/-imm8, and adjusted register including shifts. T variants used when in Privilege mode.
Load-store Single	32	2 ^b (+P ^a if PC is destination)	LDR, LDRB, LDRSB, LDRH, LDRSH, STR, STRB, and STRH, and T variants. PLD and PLI are both hints and so act as a NOP.
Load-store Multiple	32	1+N ^b (+P ^a if PC is loaded)	STM, LDM, LDRD, and STRD.
Load-store Special	32	1+N ^b	LDREX, STREX, LDREXB, LDREXH, STREXB, STREXH, CLREX. These fault if no local monitor (is IMP DEF). LDREXD and STREXD are not included in this profile.
Branches	32	1+P ^a	B, BL, and B<cond>. No BLX (1) because it always changes state. No BXJ.
System	32	1-2	MSR(2) and MRS(2) replace MSR/MRS but also do more. These access the other stacks and also the status registers. CPSIE/CPSID 32-bit forms are not supported. No RFE or SRS.
System	16	1-2	CPSIE and CPSID are quick versions of MSR(2) instructions and use the standard Thumb encodings, but only permit use of i and f and not a.
Extended32	32	1	NOP and YIELD (hinted NOP). No MRS (1), MSR (1), or SUBS (PC return link).

Table 18-1 Instruction timings (continued)

Instruction type	Size	Cycles count	Description
Combined Branch	16	1+P ^a	CBZ.
Extended	16	0-1 ^d	IT and NOP (includes YIELD).
Divide	32	2-12 ^e	SDIV and UDIV. 32/32 divides both signed and unsigned with 32-bit quotient result (no remainder, it can be derived by subtraction). This earliest out when dividend and divisor are close in size.
Sleep	32	1+W ^f	WFI, WFE, and SEV are in the class of hinted NOP instructions that control sleep behavior.
Barriers	16	1+B ^g	ISB, DSB, and DMB are barrier instructions that ensure certain actions have taken place before the next instruction is executed.
Saturation	32	1	SSAT and USAT perform saturation on a register. They perform three tasks. They normalize the value using shift, test for overflow from a selected bit position (the Q value) and set the xPSR Q bit. Saturation refers to the largest unsigned value or the largest/smallest signed value for the size selected.

- Branches take one cycle for instruction and then pipeline reload for target instruction. Non-taken branches are 1 cycle total. Taken branches with an immediate are normally 1 cycle of pipeline reload (2 cycles total). Taken branches with register operand are normally 2 cycles of pipeline reload (3 cycles total). Pipeline reload is longer when branching to unaligned 32-bit instructions in addition to accesses to slower memory. A branch hint is emitted to the code bus that permits a slower system to pre-load. This can reduce the branch target penalty for slower memory, but never less than shown here.
- Generally, load-store instructions take two cycles for the first access and one cycle for each additional access. Stores with immediate offsets take one cycle.
- UMULL/SMULL/UMLAL/SMLAL use early termination depending on the size of source values. These are interruptible (abandoned/restarted), with worst case latency of one cycle. MLAL versions take four to seven cycles and MULL versions take three to five cycles. For MLAL, the signed version is one cycle longer than the unsigned.
- IT instructions can be folded.
- DIV timings depend on dividend and divisor. DIV is interruptible (abandoned/restarted), with worst case latency of one cycle. When dividend and divisor are similar in size, divide terminates quickly. Minimum time is for cases of divisor larger than dividend and divisor of zero. A divisor of zero returns zero (not a fault), although a debug trap is available to catch this case.
- Sleep is one cycle for the instruction plus as many sleep cycles as appropriate. WFE only uses one cycle when event has passed. WFI is normally more than one cycle unless an interrupt happens to pend exactly when entering WFI.
- ISB takes one cycle (acts as branch). DMB and DSB take one cycle unless data is pending in the write buffer or LSU. If an interrupt comes in during a barrier, it is abandoned/restarted.

Cycle count information:

- P = pipeline reload
- N = count of elements

- W = sleep wait
- B = barrier clearance.

In general, each instruction takes one cycle (one core clock) to start executing as Table 18-1 on page 18-3 shows. Additional cycles can be taken because of fetch stalls.

18.3 Load-store timings

This section describes how best to pair instructions. This achieves more reductions in timing.

- STR Rx,[Ry,#imm] is always one cycle. This is because the address generation is performed in the initial cycle, and the data store is performed at the same time as the next instruction is executing. If the store is to the store buffer, and the store buffer is full, the next instruction is delayed until the store can complete. If the store is not to the store buffer, such as to the Code segment, and that transaction stalls, the impact on timing is only felt if another load or store operation is executed before completion.
- LDR Rx!,[any] is not normally pipelined. That is, base update load is generally at least a two-cycle operation (more if stalled). However, if the next instruction does not require to read from a register, the load is reduced to one cycle. Non register writing instructions include CMP, TST, NOP, and non-taken IT controlled instructions.
- LDR PC,[any] is always a blocking operation. This means minimally two cycles for the load, and three cycles for the pipeline reload. So at least five cycles (more if stalled on the load or the fetch).
- LDR Rx,[PC,#imm] might add a cycle because of contention with the fetch unit.
- TBB and TBH are also blocking operations. These are minimally two cycles for the load, one cycle for the add, and three cycles for the pipeline reload. This means at least six cycles (more if stalled on the load or the fetch).
- LDR any are pipelined when possible. This means that if the next instruction is an LDR or non-base updating STR, and the destination of the first LDR is not used to compute the address for the next instruction, then one cycle is removed from the cost of the next instruction. So, an LDR might be followed by an STR, so that the STR writes out what the LDR loaded. More multiple LDRs can be pipelined together. Some optimized examples:
 - LDR R0,[R1]; LDR R1,[R2] - normally three cycles total
 - LDR R0,[R1,R2]; STR R0,[R3,#20] - normally three cycles total
 - LDR R0,[R1,R2]; STR R1,[R3,R2] - normally three cycles total
 - LDR R0,[R1,R5]; LDR R1,[R2]; LDR R2,[R3,#4] - normally four cycles total.

- STR with register offset cannot be pipelined after. STR can only be pipelined when after an LDR, but nothing can be pipelined after the store. Even a stalled STR normally only take two cycles, because of the store buffer (bit band, data segment, and unaligned).
- LDREX and STREX can be pipelined exactly as LDR. Because STREX is treated more like an LDR, it can be pipelined as explained for LDR. Equally LDREX is treated exactly as an LDR and so can be pipelined.
- LDRD, STRD cannot be pipelined with preceding or following instructions. However, the two words are pipelined together. So, three cycles when not stalled.
- LDM, STM cannot be pipelined with preceding or following instructions. However, all elements after the first are pipelined together. So, a three element LDM takes 2+1+1 or 5 cycles when not stalled. Similarly, an eight element store takes nine cycles when not stalled. When interrupted, LDM and STM instructions continue from where left off when returned to. The continue operation adds one or two cycles to the first element to get started.
- Unaligned Word or Halfword Loads or stores add penalty cycles. A byte aligned halfword load or store adds one extra cycle to perform the operation as two bytes. A halfword aligned word load or store adds one extra cycle to perform the operation as two halfwords. A byte-aligned word load or store adds two extra cycles to perform the operation as a byte, a halfword, and a byte. These numbers increase if the memory stalls. A STR or STRH cannot delay the processor because of the store buffer.

Chapter 19

AC Characteristics

This chapter gives the timing parameters for the processor. It contains the following sections:

- *Processor timing parameters* on page 19-2.

19.1 Processor timing parameters

This section describes the input and output port timing parameters for the processor. The maximum timing parameter or constraint delay for each processor signal applied to the SoC is given as a percentage in Table 19-1 to Table 19-16 on page 19-10. The input and output delay columns provide the maximum and minimum time as a percentage of the processor clock cycle given to the SoC for that signal.

19.1.1 Input and output port timing parameters

Table 19-1 shows the timing parameters for the miscellaneous input ports.

Table 19-1 Miscellaneous input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	10%	PORESETn
Clock uncertainty	10%	SYSRESETn
Clock uncertainty	50%	BIGEND
Clock uncertainty	50%	EDBGRQ
Clock uncertainty	50%	STCLK
Clock uncertainty	50%	STCALIB[25:0]
Clock uncertainty	50%	RXEV
Clock uncertainty	50%	AUXFAULT[31:0]
Clock uncertainty	10%	IFLUSH
Clock uncertainty	50%	PPBLOCK[5:0]

Table 19-2 shows the timing parameters for the low power input ports.

Table 19-2 Low power input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	50%	SLEEPHOLDREQn
Clock uncertainty	50%	WICDSREQn

Table 19-3 shows the timing parameters for the interrupt input ports.

Table 19-3 Interrupt input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	50%	INTISR[239:0]
Clock uncertainty	50%	INTNMI
Clock uncertainty	20%	VECTADDR[9:0]
Clock uncertainty	20%	VECTADDREN

Table 19-4 shows the timing parameters for the *Advanced High-performance Bus* (AHB) ports.

Table 19-4 AHB input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	10%	DNOTITRANS
Clock uncertainty	50%	HRDATAI[31:0]
Clock uncertainty	50%	HREADYI
Clock uncertainty	50%	HRESPI[1:0]
Clock uncertainty	50%	HRDATAD[31:0]
Clock uncertainty	50%	HREADYD
Clock uncertainty	50%	HRESPD[1:0]
Clock uncertainty	50%	EXRESPD
Clock uncertainty	50%	HRDATAS[31:0]
Clock uncertainty	50%	HREADYS
Clock uncertainty	50%	HRESPS[1:0]
Clock uncertainty	50%	EXRESPS

Table 19-5 shows the timing parameter for the *Private Peripheral Bus* (PPB) port.

Table 19-5 PPB input port timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	50%	PRDATA[31:0]
Clock uncertainty	50%	PREADY
Clock uncertainty	50%	PSLVERR

Table 19-6 shows the timing parameters for the debug input ports.

Table 19-6 Debug input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	10%	nTRST
Clock uncertainty	50%	SWDITMS
Clock uncertainty	50%	TDI
Clock uncertainty	50%	DAPRESETn
Clock uncertainty	50%	DAPSEL
Clock uncertainty	50%	DAPEN
Clock uncertainty	50%	DAPENABLE
Clock uncertainty	50%	DAPCLKEN
Clock uncertainty	50%	DAPWRITE
Clock uncertainty	50%	DAPABORT
Clock uncertainty	50%	DAPADDR[31:0]
Clock uncertainty	50%	DAPWDATA[31:0]
Clock uncertainty	50%	ATREADY
Clock uncertainty	50%	DBGRESTART
Clock uncertainty	50%	FIXHMASTERTYPE

Table 19-7 shows the timing parameters for the test input ports.

Table 19-7 Test input ports timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	10%	SE
Clock uncertainty	10%	SI
Clock uncertainty	10%	RSTBYPASS
Clock uncertainty	10%	CGBYPASS
Clock uncertainty	10%	WSII
Clock uncertainty	10%	WSOI

Table 19-8 shows the timing parameters for the *Embedded Trace Macrocell* (ETM).

Table 19-8 ETM input port timing parameters

Input delay Min.	Input delay Max.	Signal name
Clock uncertainty	30%	ETMPWRUP
Clock uncertainty	50%	ETMFIFOFILL

Table 19-9 shows the timing parameters for the miscellaneous output ports.

Table 19-9 Miscellaneous output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	LOCKUP
Clock uncertainty	50%	SYSRESETREQ
Clock uncertainty	50%	BRCHSTAT[3:0]
Clock uncertainty	50%	HALTED
Clock uncertainty	50%	TXEV
Clock uncertainty	50%	ATIDITM[6:0]

Table 19-9 Miscellaneous output ports timing parameters (continued)

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	CURRPRI[7:0]
Clock uncertainty	70%	TRCENA
Clock uncertainty	50%	INTERNALSTATE

Table 19-10 shows the timing parameters for the low power output ports.

Table 19-10 Low power output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	SLEEPING
Clock uncertainty	50%	SLEEPDEEP
Clock uncertainty	50%	SLEEPHOLDACKn
Clock uncertainty	50%	WICLOAD
Clock uncertainty	50%	WICCLEAR
Clock uncertainty	50%	WICDSACKn
Clock uncertainty	50%	WICMASKNMI
Clock uncertainty	50%	WICMASKMON
Clock uncertainty	50%	WICMASKISR
Clock uncertainty	50%	WICMASKRXEV

Table 19-11 shows the timing parameters for the AHB output ports.

Table 19-11 AHB output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	HTRANSI[1:0]
Clock uncertainty	50%	HSIZEI[2:0]
Clock uncertainty	50%	HPROTI[3:0]
Clock uncertainty	50%	MEMATTRI[1:0]

Table 19-11 AHB output ports timing parameters (continued)

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	HBURSTI[2:0]
Clock uncertainty	50%	HADDRI[31:0]
Clock uncertainty	50%	HMASTERD[1:0]
Clock uncertainty	50%	HTRANSD[1:0]
Clock uncertainty	50%	HSIZED[2:0]
Clock uncertainty	50%	HPROTD[3:0]
Clock uncertainty	50%	MEMATTRD[1:0]
Clock uncertainty	50%	EXREQD
Clock uncertainty	50%	HBURSTD[2:0]
Clock uncertainty	50%	HADDRD[31:0]
Clock uncertainty	50%	HWDATAD[31:0]
Clock uncertainty	50%	HWRITED
Clock uncertainty	50%	HMASTERS[1:0]
Clock uncertainty	50%	HTRANS[1:0]
Clock uncertainty	50%	HSIZES[2:0]
Clock uncertainty	50%	HPROTS[3:0]
Clock uncertainty	50%	MEMATTRS[1:0]
Clock uncertainty	50%	EXREQS
Clock uncertainty	50%	HBURSTS[2:0]
Clock uncertainty	50%	HMASTLOCKS
Clock uncertainty	50%	HADDRS[31:0]
Clock uncertainty	50%	HWDATAS[31:0]
Clock uncertainty	50%	HWRITES

Table 19-12 shows the timing parameters for the PPB output ports.

Table 19-12 PPB output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	PADDR31
Clock uncertainty	50%	PADDR[19:2]
Clock uncertainty	50%	PSEL
Clock uncertainty	50%	PENABLE
Clock uncertainty	50%	PWRITE
Clock uncertainty	50%	PWDATA[31:0]

Table 19-13 shows the timing parameters for the debug interface output ports.

Table 19-13 Debug interface output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	SWV
Clock uncertainty	50%	TRACECLK
Clock uncertainty	50%	TRACEDATA[3:0]
Clock uncertainty	50%	TDO
Clock uncertainty	50%	SWDO
Clock uncertainty	50%	nTDOEN
Clock uncertainty	50%	SWDOEN
Clock uncertainty	50%	DAPREADY
Clock uncertainty	50%	DAPSLVERR
Clock uncertainty	50%	DAPRDATA[31:0]
Clock uncertainty	50%	ATVALID
Clock uncertainty	50%	AFREADY
Clock uncertainty	50%	ATDATA[7:0]
Clock uncertainty	50%	DBGRESTARTED

Table 19-14 shows the timing parameters for the ETM interface output ports.

Table 19-14 ETM interface output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	30%	ETMTRIGGER[3:0]
Clock uncertainty	30%	ETMTRIGINOTD[3:0]
Clock uncertainty	30%	ETMIVALID
Clock uncertainty	30%	ETMDVALID
Clock uncertainty	30%	ETMFOLD
Clock uncertainty	30%	ETMCANCEL
Clock uncertainty	30%	ETMIA[31:1]
Clock uncertainty	30%	ETMICCFail
Clock uncertainty	30%	ETMIBRANCH
Clock uncertainty	30%	ETMIINDBR
Clock uncertainty	30%	ETMFLUSH
Clock uncertainty	30%	ETMFINDBR
Clock uncertainty	30%	ETMINTSTAT[2:0]
Clock uncertainty	30%	ETMINTNUM[8:0]
Clock uncertainty	30%	ETMISTALL
Clock uncertainty	30%	DSYNC

Table 19-15 shows the timing parameters for the *AHB Trace Macrocell* (HTM) interface output ports.

Table 19-15 HTM interface output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	HTMDHADDR[31:0]
Clock uncertainty	50%	HTMDHTRANS[1:0]
Clock uncertainty	50%	HTMDHSIZE[2:0]

Table 19-15 HTM interface output ports timing parameters (continued)

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	50%	HTMDHBURST[2:0]
Clock uncertainty	50%	HTMDHPROT[3:0]
Clock uncertainty	50%	HTMDHWDATA[31:0]
Clock uncertainty	50%	HTMDHWRITE
Clock uncertainty	50%	HTMDHRDATA[31:0]
Clock uncertainty	50%	HTMDHREADY
Clock uncertainty	50%	HTMDHRESP[1:0]

Table 19-16 shows the timing parameters for the test output ports.

Table 19-16 Test output ports timing parameters

Output delay Min.	Output delay Max.	Signal name
Clock uncertainty	10%	SO
Clock uncertainty	10%	WSOO
Clock uncertainty	10%	WSIO

Appendix A

Signal Descriptions

This appendix lists and describes the processor interface signals. It contains the following sections:

- *Clocks* on page A-2
- *Resets* on page A-3
- *Miscellaneous* on page A-4
- *Interrupt interface signals* on page A-6
- *Low power interface* on page A-7
- *ICode interface* on page A-8
- *DCode interface* on page A-9
- *System bus interface* on page A-10
- *Private Peripheral Bus interface* on page A-11
- *ITM interface* on page A-12
- *AHB-AP interface* on page A-13
- *ETM interface* on page A-14
- *AHB Trace Macrocell interface* on page A-16
- *Test interface* on page A-17
- *WIC interface* on page A-18.

A.1 Clocks

Table A-1 lists the clock signals.

Table A-1 Clock signals

Name	Direction	Description
HCLK	Input	Main Cortex-M3 clock
FCLK	Input	Free-running Cortex-M3 clock
DAPCLK	Input	AHB-AP clock

A.2 Resets

Table A-2 lists the reset signals.

Table A-2 Reset signals

Name	Direction	Description
PORESETn	Input	Power-on reset. Resets entire Cortex-M3 system.
SYSRESETn	Input	System reset. Resets processor, non-debug portion of NVIC, Bus Matrix, and MPU. Debug components are not reset.
SYSRESETREQ	Output	System reset request.
DAPRESETn	Input	AHB-AP reset.

A.3 Miscellaneous

Table A-3 lists the leftover signals.

Table A-3 Miscellaneous signals

Name	Direction	Description
LOCKUP	Output	LOCKUP gives immediate indication of seriously errant kernel software. This is the result of the core being locked up because of an unrecoverable exception following the activation of the processor's built in system state protection hardware. For more information about the ARMv7-M architectural lock up conditions see the ARMv7-M Architecture Reference Manual.
CURRPRI[7:0]	Output	Indicates what priority interrupt (or base boost) is currently used. CURRPRI represents the pre-emption priority, and does not indicate the secondary priority.
HALTED	Output	In halting debug mode. HALTED remains asserted while the core is in debug.
DBGRESTARTED	Output	Handshake for DBGRESTART .
TXEV	Output	Event transmitted as a result of SEV instruction. This is a single cycle pulse.
TRCENA	Output	Trace Enable. This signal reflects the setting of bit [24] of the Debug Exception and Monitor Control Register. This signal gate the clock to the TPIU and ETM blocks to reduce power consumption when trace is disabled.
INTERNALSTATE[148:0]	Output	Internal state.
BIGEND	Input	Static endian select: 1 = big-endian 0 = little-endian This signal is sampled at reset, and cannot be changed when reset is inactive.
EDBGRQ	Input	External debug request.
PPBLOCK[5:0]	Input	Reserved. Must be tied to 6'b000000.
STCLK	Input	System Tick Clock.
STCALIB[25:0]	Input	System Tick Calibration.
RXEV	Input	Causes a wakeup from a WFE instruction.
VECTADDR[9:0]	Input	Reserved. Must be tied to 10'b0000000000.

Table A-3 Miscellaneous signals (continued)

Name	Direction	Description
VECTADDREN	Input	Reserved. Must be tied to 1'b0.
DNOTITRANS	Input	Static tie-off that forces the processor to not permit ICode and DCode AHB transactions to occur at the same time. This permits a simple bus multiplexer to be instantiated externally to the processor.
AUXFAULT[31:0]	Input	Auxiliary fault status information from the system.
IFLUSH	Input	Reserved. Instruction flush, must be tied to 0.
DBGRESTART	Input	External restart request.

A.4 Interrupt interface

Table A-4 lists the signals of the external interrupt interface.

Table A-4 Interrupt interface signals		
Name	Direction	Description
INTISR[239:0]	Input	External interrupt signals
INTNMI	Input	Non-maskable interrupt

A.5 Low power interface

Table A-5 lists the signals of the low power interface.

Table A-5 Low power interface signals

Name	Direction	Description
SLEEPDEEP	Output	Indicates that the Cortex-M3 clock can be stopped.
SLEEPING	Output	Indicates that the Cortex-M3 clock can be stopped.
SLEEPHOLDACKn	Output	Acknowledge signal for SLEEPHOLDREQn .
WICMASKISR	Output	WIC Active high set of signals indicating which interrupts would cause wakeup.
WICMASKMON	Output	WIC Active high signal indicating that debug monitor would cause wakeup.
WICMASKNMI	Output	WIC Active high signal indicating that NMI would cause wakeup.
WICMASKRXEV	Output	WIC Active high signal indicating that RXEV would cause wakeup.
WICLOAD	Output	Causes WIC to be loaded with sensitivity data given by WICMASK* .
WICCLEAR	Output	Causes WIC to clear registered sensitivity data.
WICDSACKn	Output	Active low indication that accepts WICDSREQn and means that SLEEPDEEP is WIC mode sleep.
SLEEPHOLDREQn	Input	Request to extend sleep. Can only be asserted when SLEEPING is high.
WICDSREQn	Input	WIC Active low request from WIC that SLEEPDEEP be WIC mode sleep.

A.6 ICode interface

Table A-6 lists the signals of the ICode interface.

Table A-6 ICode interface

Name	Direction	Description
HADDRI[31:0]	Output	32-bit instruction address bus
HTRANSI[1:0]	Output	Indicates whether the current transfer is IDLE or NONSEQUENTIAL.
HSIZEI[2:0]	Output	Indicates the size of the instruction fetch. All instruction fetches are 32-bit on Cortex-M3.
HBURSTI[2:0]	Output	Indicates if the transfer is part of a burst. All instruction fetches and vector table loads are performed as SINGLE on Cortex-M3.
HPROTI[3:0]	Output	Provides information on the access. Always indicates cacheable and non-bufferable on this bus. HPROTI[0] = 0 indicates instruction fetch HPROTI[0] = 1 indicates vector fetch
MEMATTRI[1:0]	Output	Memory attributes. Always 01 for this bus (non-shareable, allocate).
BRCHSTAT[3:0]	Output	Provides hint information on the current or coming AHB fetch requests. Conditional opcodes could be a speculation and subsequently discarded. 0000 No hint 0001 Conditional branch backwards in decode 0010 Conditional branch in decode 0011 Conditional branch in execute 0100 Unconditional branch in decode 0101 Unconditional branch in execute 0110 Reserved 0111 Reserved 1000 Conditional branch in decode taken (cycle after IHTRANS) 1001 ... 1111 Reserved
HRDATAI[31:0]	Input	Instruction read bus.
HREADYI	Input	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.
HRESPI[1:0]	Input	The transfer response status. OKAY or ERROR.

A.7 DCode interface

Table A-7 lists the signals of the DCode interface.

Table A-7 DCode interface

Name	Direction	Description
HADDRD[31:0]	Output	32-bit data address bus
HTRANSD[1:0]	Output	Indicates whether the current transfer is IDLE, NONSEQUENTIAL, or SEQUENTIAL.
HWRITED	Output	Write not read
HSIZED[2:0]	Output	Indicates the size of the access. Can be 8, 16, or 32 bits.
HBURSTD[2:0]	Output	Indicates if the transfer is part of a burst. Data accesses are performed as INCR on Cortex-M3.
HPROTD[3:0]	Output	Provides information on the access. Always indicates cacheable and non-bufferable on this bus.
EXREQD	Output	Exclusive request.
MEMATTRD[1:0]	Output	Memory attributes. Always 01 for this bus (non-shareable, allocate).
HMASTERD[1:0]	Output	Indicates the current DCode bus master: <ul style="list-style-type: none"> 0 = Core data side accesses. 1 = DAP accesses. 2 = Core instruction side accesses. These include vector fetches that are marked as data by HPROTD[0]. This value cannot appear on HMASTERD. 3 = Reserved. This value cannot appear on HMASTERD.
HWDATAD[31:0]	Output	32-bit write data bus.
HREADYD	Input	When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.
HRESPD[1:0]	Input	The transfer response status. OKAY or ERROR.
HRDATAD[31:0]	Input	Read data.
EXRESPD	Input	Exclusive response.

A.8 System bus interface

Table A-8 lists the signals of the system bus interface.

Table A-8 System bus interface

Name	Direction	Description
HADDRS[31:0]	Output	32-bit address.
HTRANS[1:0]	Output	Indicates the type of the current transfer. Can be IDLE, NONSEQUENTIAL, OR SEQUENTIAL.
HSIZES[2:0]	Output	Indicates the size of the access. Can be 8, 16, or 32 bits.
HBURSTS[2:0]	Output	Indicates if the transfer is part of a burst.
HPROTS[3:0]	Output	Provides information on the access.
HWDATAS[31:0]	Output	32-bit write data bus.
HWRITES	Output	Write not read.
HMASTLOCKS	Output	Indicates a transaction that must be atomic on the bus. This is only for bit-band writes (performed as read-modify-write).
EXREQS	Output	Exclusive request.
MEMATTRS[1:0]	Output	Memory attributes. Bit 0 = Allocate, Bit 1 = shareable.
HMASTERS[1:0]	Output	Indicates the current system bus master: <ul style="list-style-type: none">0 = Core data side accesses or DAP access with MasterType set to 0.1 = DAP accesses with MasterType set to 1.2 = Core instruction side accesses. These include vector fetches that are marked as data by HPROTS[0].3 = Reserved. This value cannot appear on HMASTERS.
HRDATAS[31:0]	Input	Read data bus.
HREADYS	Input	When HIGH indicates that a transfer has completed on the bus. The signal is driven LOW to extend a transfer.
HRESPS[1:0]	Input	The transfer response status. OKAY or ERROR.
EXRESPS	Input	Exclusive response.

A.9 Private Peripheral Bus interface

Table A-9 lists the signals of the PPB interface.

Table A-9 Private Peripheral Bus interface

Name	Direction	Description
PADDR[19:2]	Output	17-bit address. Only the bits that are relevant to the External Private Peripheral Bus are driven.
PADDR31	Output	This signal is driven HIGH when the AHB-AP is the requesting master. It is driven LOW when DCore is the requesting master.
PSEL	Output	Indicates that a data transfer is requested.
PENABLE	Output	Strobe to time all accesses. Indicates the second cycle of an APB transfer.
PWDATA[31:0]	Output	32-bit write data bus.
PWRITE	Output	Write not read.
PRDATA[31:0]	Input	Read data bus.
PREADY	Input	APB slave ready.
PSLVERR	Input	APB slave error.

A.10 ITM interface

Table A-10 lists the signals of the ITM interface.

Table A-10 ITM interface

Name	Direction	Description
ATVALID	Output	ATB valid.
AFREADY	Output	ATB flush.
ATDATA[7:0]	Output	ATB data.
ATIDITM[6:0]	Output	ITM ID for TPIU.
ATREADY	Input	ATB ready.
TPIUACTV	Input	TPIU active indication signal.
TPIUBAUD	Input	Reference for the timestamp counter, so that timestamps are at the observable baud rate of the external protocol.

A.11 AHB-AP interface

Table A-11 lists the signals of the AHB-AP interface.

Table A-11 AHB-AP interface

Name	Direction	Description
DAPRDATA[31:0]	Output	The read bus is driven by the selected AHB-AP during read cycles when DAPWRITE is LOW.
DAPREADY	Output	The AHB-AP uses this signal to extend a DAP transfer.
DAPSLVERR	Output	The error response is because of: <ul style="list-style-type: none"> Master port produced an error response, or transfer not initiated because of DAPEN preventing a transfer. Access to AP register not accepted after a DAPABORT operation.
DAPCLKEN	Input	DAP clock enable (power saving).
DAPEN	Input	AHB-AP enable.
DAPADDR[31:0]	Input	DAP address bus.
DAPSEL	Input	Select signal generated from the DAP decoder to each AP. This signal indicates that the slave device is selected, and a data transfer is required. There is a DAPSEL signal for each slave. The signal is not generated by the driving DP. The decoder monitors the address bus and asserts the relevant DAPSEL .
DAPENABLE	Input	This signal indicates the second and subsequent cycles of a DAP transfer from DP to AHB-AP.
DAPWRITE	Input	When HIGH indicates a DAP write access from DP to AHB-AP. When LOW indicates a read access.
DAPWDATA[31:0]	Input	The write bus is driven by the DP block during write cycles when DAPWRITE is HIGH.
DAPABORT	Input	Aborts the current transfer. The AHB-AP returns DAPREADY HIGH without affecting the state of the transfer in progress in the AHB Master Port.
FIXMASTERTYPE	Input	Setting this signal to 1 overrides the MasterType bit of the AHB-AP <i>Control and Status Word</i> (CSW). This ensures accesses from the debugger is always issued as 0x1 on HMASTERD and HMASTERS regardless of the MasterType setting in the CSW.

A.12 ETM interface

Table A-12 lists the signals of the ETM interface.

Table A-12 ETM interface

Name	Direction	Description
ETMTRIGGER[3:0]	Output	Trigger from DWT. One bit for each of the four DWT comparators.
ETMTRIGINOTD[3:0]	Output	Indicates if the ETM is triggered on an instruction or data match.
ETMIVALID	Output	Instruction valid.
ETMIA[31:1]	Output	PC of the instruction being executed.
ETMICCFAIL	Output	Condition Code fail. Indicates if the current instruction has failed or passed its conditional execution check.
ETMIBRANCH	Output	Opcode is a branch target.
ETMIINDBR	Output	Opcode is an indirect branch target.
ETMINTSTAT[2:0]	Output	Interrupt status. Marks interrupt status of current cycle. 000 - no status 001 - interrupt entry 010 - interrupt exit 011 - interrupt return 100 - vector fetch and stack push. ETMINTSTAT entry/return is asserted in the first cycle of the new interrupt context. Exit occurs without ETMIVALID .
ETMINTNUM[8:0]	Output	Marks the interrupt number of the current execution context.
ETMISTALL	Output	Indicates that the last instruction signalled by the core has not yet entered execute.
ETMFLUSH	Output	A PC modifying opcode has executed, or an interrupt push/pop has started.
ETMPWRUP	Input	ETM is enabled
ETMDVALID	Output	Data valid
ETMCANCEL	Output	Instruction cancelled
ETMFINDBR	Output	Flush is indirect. Marks flush hint destination cannot be inferred from the PC.

Table A-12 ETM interface (continued)

Name	Direction	Description
ETMFOLD	Output	Opcode fold. An IT opcode has been folded in this cycle. PC advances past the current (16-bit) opcode plus the IT instruction (16 bits). This is reflected in the ETMIA.
ETMFIFOFULL	Input	Driven by the ETM (if connected). ETMFIFOFULL is asserted when the ETM FIFO is full, and causes the processor to stall until the FIFO has drained, so ensuring that no trace is lost.
DSYNC	Output	Synchronization pulse from DWT.

A.13 AHB Trace Macrocell interface

Table A-13 lists the signals of the *AHB Trace Macrocell* (HTM) interface

Table A-13 HTM interface

Name	Direction	Description
HTMDHADDR[31:0]	Output	32-bit address
HTMDHTRANS[1:0]	Output	Output indicates the type of the current data transfer. Can be IDLE, NONSEQUENTIAL, OR SEQUENTIAL.
HTMDHSIZE[1:0]	Output	Indicates the size of the access. Can be 8, 16, or 32 bits.
HTMDHBURST[2:0]	Output	Output indicates if the transfer is part of a burst.
HTMDHPROT[3:0]	Output	Provides information on the access.
HTMDHWDATA[31:0]	Output	32-bit write data bus.
HTMDHWRITE	Output	Write not read.
HTMDHRDATA[31:0]	Output	Read data bus.
HTMDHREADY	Output	Ready signal.
HTMDHRESP[1:0]	Output	The transfer response status. OKAY or ERROR.

A.14 Test interface

Table A-14 lists the signals of the test interface.

Table A-14 Test interface

Name	Direction	Description
SE	Input	Scan enable.
RSTBYPASS	Input	Reset bypass for scan testing. PORESETn is the only reset used during scan testing.
CGBYPASS	Input	Architectural clock gate bypass for scan testing.

A.15 WIC interface

Table A-15 lists the signals of the WIC interface.

Table A-15 WIC interface signals

Name	Direction	Description
WAKEUP	Output	Active high signal to PMU that core must be made active.
WICSENSE	Output	Active high set of signals indicating which input lines the WIC would generate WAKEUP signal in response to.
WICPEND	Output	Captured interrupt information for NVIC.
WICENACK	Output	Active high SLEEPDEEP is WICSLEEP acknowledgement to PMU.
WICDSREQn	Output	Active low request to NVIC to make SLEEPDEEP mode WIC-sleep.
FCLK	Input	Clock synchronous to NVIC FCLK input.
nRESET	Input	Asynchronous active low reset.
WICDISABLE	Input	Debugger active signal to disable WIC mode when a debugger is attached.
WICINT	Input	Peripherals Active high interrupt, debug monitor, NMI , and or RXEV signals.
WICMASK	Input	Active high set of signals indicating which input lines the WIC should generate a WAKEUP signal in response to.
WICLOAD	Input	Load interrupt sensitivity list into WIC from NVIC
WICCLEAR	Input	Clear sensitivity list in WIC.
WICENREQ	Input	Make SLEEPDEEP mode WIC mode sleep request from PMU.
WICDSACKn	Input	Active low SLEEPDEEP is WICSLEEP acknowledgement from NVIC.

Appendix B

Revisions

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

Table B-1 lists the differences between issue E and issue F, and Table B-2 on page B-5 lists the differences between issue F and issue G.

Table B-1 Differences between issue E and issue F

Change	Location
Introductory processor information updated	<i>About the processor</i> on page 1-2
Processor block diagram updated	Figure 1-1 on page 1-5
Introductory information added, including: <ul style="list-style-type: none">• TPIU subsection• Addition of note to SW/SWJ-DP subsection• ROM table subsection.	
Introductory processor core information updated	<i>Processor core</i> on page 1-5
APB bus now version 3.0	<i>Bus matrix</i> on page 1-7

Table B-1 Differences between issue E and issue F (continued)

Change	Location
Configurable options information expanded to include: <ul style="list-style-type: none"> Added DWT configurability information New subsections for ITM, AHB-AP, FPB and Observation 	
New subsection added to list changes in functionality between r1p1 and r2p0	<i>Differences in functionality between r1p1 and r2p0</i> on page 1-20
Information about the programmer's model updated	<i>About the programmer's model</i> on page 2-2
Definition of ICI field of Execution Program Status Register updated	Table 2-3 on page 2-8
Table of nonsupported Thumb instructions removed.	
Second footnote on Table 5-1 removed.	Table 5-1 on page 5-4
Addition of note to vector table and reset description	<i>Vector Table and Reset</i> on page 5-20
Description of SLEEPING and SLEEPDEEP signals updated.	<i>System power management</i> on page 7-3
Description of extending sleep functionality added	<i>Extending sleep</i> on page 7-5
Addition of Auxiliary Control Register	Table 8-1 on page 8-3 and <i>NVIC register descriptions</i> on page 8-7
Irq 0 to 31 Priority Register amended to Irq 0 to 3 Priority Register	Table 8-1 on page 8-3
Irq 236 to 239 Priority Register amended to Irq 224 to 239 Priority Register	Table 8-1 on page 8-3
HCLK changed to FCLK	<i>Level versus pulse interrupts</i> on page 8-43
Addition of ascending MPU region priority information	<i>About the MPU</i> on page 9-2
Extra paragraph added.	<i>About core debug</i> on page 10-2
Debug Core Register Selector Register REGSEL bit field function updated	Table 10-3 on page 10-7
Extra paragraph added.	<i>About system debug</i> on page 11-2
Paragraph added about removing FPB	<i>FPB</i> on page 11-6

Table B-1 Differences between issue E and issue F (continued)

Change	Location
Addition of note about configuring flash patch registers to be present or not	<i>FPB programmer's model</i> on page 11-6
First bullet point updated	<i>DWT</i> on page 11-13
Addition of note about configuring DWT registers to be present or not	<i>Summary and description of the DWT registers</i> on page 11-13
DWT Control Register reset state updated	<i>DWT Control Register</i> on page 11-15
DWT Control Register bit assignments updated	Figure 11-5 on page 11-16 and Table 11-7 on page 11-16
Addition of note about configuring ITM registers to be present or not	<i>Summary and description of the ITM registers</i> on page 11-30
ITM Trace Control Register TSENA field bit function updated	Table 11-22 on page 11-34
Addition of note about configuring AHB-AP registers to be present or not	<i>Summary and description of the AHB-AP registers</i> on page 11-39
AHB-AP Banked Data Register DATA field reset value removed	Table 11-32 on page 11-43
Addition of information about absence of debug functionality	<i>About the DP</i> on page 13-2
Information about exclusive memory accesses updated	<i>Exclusives</i> on page 12-6 and <i>Exclusives</i> on page 12-7
Note about bit-band accesses updated	<i>Bit-band accesses</i> on page 12-13
ETM block diagram updated	Figure 14-1 on page 14-3
HCLK and CLK replaced by FCLK	Table 14-2 on page 14-4, Table 14-3 on page 14-5, Table 14-4 on page 14-5, Table 14-5 on page 14-6, and Table 14-6 on page 14-6

Table B-1 Differences between issue E and issue F (continued)

Change	Location
ETM Trigger Even Register description upgraded	Table 14-9 on page 14-16
ETM Status Register description updated	
TraceEnable register replaced by Trace Start/Stop Resource Control	
TraceEnable Control 2 register added	
Lock Status Register added	
Description of FIFOFULL Region Register added	
Description of FIFOFULL Level Register updated	
Description of CoreSight Trace ID Register updated	<i>ETM Control Register</i> on page 14-19
Description ETM Control Register implementation bits expanded	
Description of TraceEnable Control 1 Register updated	
Description ETM ID Register updated to reflect revision 2	
Subsection describing ETM Event Resources added	
Subsection describing Cross Trigger Interface added	
Branch status interface section updated	
Note about HADDRICore and HTRANSICore removed	Figure 15-9 on page 15-13
Example of an opcode sequence timing diagram updated	
Description of APB interface inputs added	

Table B-1 Differences between issue E and issue F (continued)

Change	Location
Addition of note about configuring TPIU registers to be present or not	<i>Summary of the TPIU registers</i> on page 17-8
The following TPIU registers removed from summary table and descriptions: <ul style="list-style-type: none"> • Trigger control registers • EXTCTL port registers • Test pattern registers 	Table 17-5 on page 17-8 and <i>Description of the TPIU registers</i> on page 17-9
The following TPIU registers added to the summary table and descriptions: <ul style="list-style-type: none"> • Integration Register: TRIGGER • Integration Mode Control Register • Integration Register: FIFO data 0 • Integration Register: FIFO data 1 • Claim tag set register • Claim tag clear register • Device ID register • PID registers • CID registers 	

Table B-2 Differences between issue F and issue G

Change	Location
<i>Wake-up Interrupt Controller (WIC)</i> added to Cortex-M3 block diagram	Figure 1-1 on page 1-5
Section 1-2 and section 1-3 combined	<i>Components, hierarchy, and implementation</i> on page 1-4
New subsection added to list changes in functionality between r1p1 and r2p0	<i>Differences in functionality between r1p1 and r2p0</i> on page 1-20
New subsection added to describe the WIC	<i>WIC</i> on page 1-10
New bullet point to describe FIXHMASTERTYPE pin	<i>Differences in functionality between r1p1 and r2p0</i> on page 1-20
Table of supported instruction removed	<i>Chapter 2 Programmer's Model</i>

Table B-2 Differences between issue F and issue G (continued)

Change	Location
More information added about the stacked xPSR	<i>Saved xPSR bits</i> on page 2-9
Reset value of Configuration Control Register changed to 0x00000200	Table 3-1 on page 3-2
System and Vendor_SYS memory regions added to table of memory region permissions	Table 4-2 on page 4-4
Memory region for Private Peripheral Bus changed to +0000000	
SLEEPHOLDREQ changed to SLEEPHOLDREQn	Throughout the book
SLEEPHOLDACK changed to SLEEPHOLDACKn	Throughout the book
DEEPSLEEP signal changed to SLEEPDEEP	Throughout the book
DBGRESTARTACK changed to DBGRESTARTED	Throughout the book
DBGRESTARTREQ changed to DBGRESTART	Throughout the book
New subsection added to describe the WIC	<i>Using the Wake-up Interrupt Controller</i> on page 7-6
Address of Irq 224 to 239 Priority Register changed to 0xE000E4EC	Table 8-1 on page 8-3
Enhanced description of function of C_MASKINTS field	Table 10-2 on page 10-4
Settings for DWT Function Registers updated	Table 11-18 on page 11-28
Minor change to timing information of ETMIA	Figure 15-4 on page 15-9
Change to timing information for ETMIVALID	Figure 15-7 on page 15-11
SLEEPHOLDREQn removed from table of miscellaneous input ports timing parameters	Table 19-1 on page 19-2
Table of low power input ports timing parameters added	Table 19-2 on page 19-2
FIXHMASTERTYPE added to table of debug input ports timing parameters	Table 19-6 on page 19-4
Input changed to Output in table header	Table 19-9 on page 19-5, Table 19-11 on page 19-6 and Table 19-12 on page 19-8 to Table 19-16 on page 19-10 inclusive
SLEEPING, SLEEPDEEP, and SLEEPHOLDACKn removed from table of miscellaneous output ports timing parameters	<i>Miscellaneous output ports timing parameters</i> on page 19-5

Table B-2 Differences between issue F and issue G (continued)

Change	Location
SLEEPDEEP , SLEEPING , SLEEPHOLDREQ , and SLEEPHOLDACK removed	Table A-3 on page A-4
New section added to describe the low power interface signals	<i>Low power interface signals</i> on page A-7
New section added to describe the WIC interface signals	<i>WIC interface signals</i> on page A-18
SLEEPHOLDACKn removed from table of miscellaneous signals	Table A-3 on page A-4
Asserted changed to de-asserted in the description of SLEEPHOLDREQn in table of low power interface signals	Table A-5 on page A-7
FIXMASTERTPYE added to list of AHB-AP interface signals	Table A-11 on page A-13

Glossary

This glossary describes some of the terms used in technical documents from ARM.

Abort

A mechanism that indicates to a core that the attempted memory access is invalid or not allowed or that the data returned by the memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid or protected instruction or data memory.

See also Data Abort, External Abort and Prefetch Abort.

Addressing modes

Various mechanisms, shared by many different instructions, for generating values used by the instructions.

Advanced High-performance Bus (AHB)

A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)

A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. 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.

Advanced Peripheral Bus (APB)

A simpler bus protocol than AXI and 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.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)

An optional component of the DAP that provides an AHB interface to a SoC.

AHB-AP *See* AHB Access Port.

AHB-Lite A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

AHB Trace Macrocell

A hardware macrocell that, when connected to a processor core, outputs data trace information on a trace port.

Aligned A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. 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.

AMBA *See* Advanced Microcontroller Bus Architecture.

Advanced Trace Bus (ATB)

A bus used by trace devices to share CoreSight capture resources.

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.

Application Specific Standard Part/Product (ASSP)

An integrated circuit that has been designed to perform a specific application function. Usually consists of two or more separate circuit functions combined as a building block suitable for use in a range of products for one or more specific application markets.

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, ARMv7-M architecture.

ARM instruction

An instruction of the ARM Instruction Set Architecture (ISA). These cannot be executed by the Cortex-M3.

ARM state

The processor state in which the processor executes the instructions of the ARM ISA. The processor only operates in Thumb state, never in ARM state.

ASIC

See Application Specific Integrated Circuit.

ASSP

See Application Specific Standard Part/Product.

ATB

See Advanced Trace Bus.

ATB bridge

A synchronous ATB bridge provides a register slice to facilitate timing closure through the addition of a pipeline stage. It also provides a unidirectional link between two synchronous ATB domains.

An asynchronous ATB bridge provides a unidirectional link between two ATB domains with asynchronous clocks. It is intended to support connection of components with ATB ports residing in different clock domains.

Base register

A register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory.

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 data transfer within a burst. For example, an INCR4 burst comprises four beats.

BE-8

Big-endian view of memory in a byte-invariant system.

See also BE-32, LE, Byte-invariant and Word-invariant.

BE-32	Big-endian view of memory in a word-invariant system. <i>See also</i> BE-8, LE, Byte-invariant and Word-invariant.
Big-endian	Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory. <i>See also</i> Little-endian and Endianness.
Big-endian memory	Memory in which: <ul style="list-style-type: none">• 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. <i>See also</i> 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.
Branch folding	Branch folding is a technique where the branch instruction is completely removed from the instruction stream presented to the execution pipeline.
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. <i>See also</i> 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 AMBA are controlled using signals to indicate the length of the burst and how the addresses are incremented. <i>See also</i> Beat.
Byte	An 8-bit data item.

Byte-invariant	<p>In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. The ARM architecture supports byte-invariant systems in ARMv6 and later versions. When byte-invariant support is selected, unaligned halfword and word memory accesses are also supported. Multi-word accesses are expected to be word-aligned.</p> <p><i>See also</i> Word-invariant.</p>
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.
Clocks Per Instruction (CPI)	<i>See</i> Cycles Per Instruction (CPI).
Cold reset	<p>Also known as power-on reset.</p> <p><i>See also</i> Warm reset.</p>
Context	<p>The environment that each process operates in for a multitasking operating system.</p> <p><i>See also</i> Fast context switch.</p>
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	<i>See</i> Warm reset.
CoreSight	The infrastructure for monitoring, tracing, and debugging a complete system on chip.
CPI	<i>See</i> Cycles per instruction.
Cycles Per instruction (CPI)	Cycles per instruction (or clocks per instruction) is a measure of the number of computer instructions that can be performed in one clock cycle. This figure of merit can be used to compare the performance of different CPUs that implement the same instruction set against each other. The lower the value, the better the performance.
Data Abort	<p>An indication from a memory system to the core of an attempt to access an illegal data memory location. An exception must be taken if the processor attempts to use the data that caused the abort.</p> <p><i>See also</i> Abort.</p>
DCode Memory	Memory space at 0x00000000 to 0x1FFFFFFF.

Debug Access Port (DAP)

A TAP block that acts as an AMBA, AHB or AHB-Lite, master for access to a system bus. The DAP is the term used to encompass a set of modular blocks that support system wide debug. The DAP is a modular component, intended to be extendable to support optional access to multiple systems such as memory mapped AHB and CoreSight APB through a single debug interface.

Debugger

A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Embedded Trace Macrocell (ETM)

A hardware macrocell that, when connected to a processor core, outputs instruction trace information on a trace port.

Endianness

Byte ordering. The scheme that determines the order that 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

An error or event which can cause the processor to suspend the currently executing instruction stream and execute a specific exception handler or interrupt service routine. The exception could be an external interrupt or NMI, or it could be 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 service routine to deal with the exception.

Exception handler

See Interrupt service routine.

Exception vector

See Interrupt vector.

External PPB

PPB memory space at 0xE0040000 to 0xE00FFFFF.

Flash Patch and Breakpoint unit (FPB)

A set of address matching tags, that reroute accesses into flash to a special part of SRAM. This permits patching flash locations for breakpointing and quick fixes or changes.

Formatter

The formatter is an internal input block in the ETB and TPIU that embeds the trace source ID within the data to create a single trace stream.

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. <i>See also</i> Monitor debug-mode.
Host	A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.
HTM	<i>See</i> AHB Trace Macrocell.
ICode Memory	Memory space at 0x00000000 to 0x1FFFFFFF.
Illegal instruction	An instruction that is architecturally Undefined.
Implementation-defined	The behavior is not architecturally defined, but is defined and documented by individual implementations.
Implementation-specific	The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.
Instruction cycle count	The number of cycles for which an instruction occupies the Execute stage of the pipeline.
Instrumentation trace	A component for debugging real-time systems through a simple memory-mapped trace interface, providing printf style debugging.
Intelligent Energy Management (IEM)	A technology that enables dynamic voltage scaling and clock frequency variation to be used to reduce power consumption in a device.
Internal PPB	PPB memory space at 0xE0000000 to 0xE003FFFF.
Interrupt service routine	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 that contains the first instruction of the corresponding interrupt service routine.
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 Debug Port (JTAG-DP)

An optional external interface for the DAP that provides a standard JTAG interface for debug access.

JTAG-DP *See* JTAG Debug Port.

LE Little endian view of memory in both byte-invariant and word-invariant systems. *See* also Byte-invariant, Word-invariant.

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.

Memory coherency A memory is coherent if the value read by a data read or instruction fetch is the value that was most recently written to that location. Memory coherency is made difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer and a cache.

Memory Protection Unit (MPU)

Hardware that controls access permissions to blocks of memory. Unlike an MMU, an MPU does not modify addresses.

Microprocessor	<i>See</i> Processor.
Monitor debug-mode	<p>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.</p> <p><i>See also</i> Halt mode.</p>
MPU	<i>See</i> Memory Protection Unit.
Multi-layer	An interconnect scheme similar to a cross-bar switch. Each master on the interconnect has a direct link to each slave, The link is not shared with other masters. This enables each master to process transfers in parallel with other masters. Contention only occurs in a multi-layer interconnect at a payload destination, typically the slave.
Nested Vectored Interrupt Controller (NVIC)	Provides the processor with configurable interrupt handling abilities.
NVIC	<i>See</i> Nested Vectored Interrupt Controller.
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.
PFU	<i>See</i> Prefetch Unit.
PMU	<i>See</i> Power Management Unit.
Power Management Unit (PMU)	Provides the processor with power management capability.
Power-on reset	<i>See</i> Cold reset.
PPB	<i>See</i> Private Peripheral Bus.
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	<p>An indication from a memory system to the core that an instruction has been fetched from an illegal memory location. An exception must be taken if the processor attempts to execute the instruction. A Prefetch Abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.</p> <p><i>See also</i> Data Abort, Abort.</p>
Prefetch Unit (PFU)	The PFU fetches instructions from the memory system that can supply one word each cycle. The PFU buffers up to three word fetches in its FIFO, which means that it can buffer up to three 32-bit Thumb instructions or six 16-bit Thumb instructions.

Private Peripheral Bus

Memory space at 0xE0000000 to 0xE00FFFFF.

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.

RealView ICE

A system for debugging embedded processor cores using a JTAG interface.

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.

SBO

See Should Be One.

SBZ

See Should Be Zero.

SBZP

See Should Be Zero or Preserved.

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.

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.

Serial-Wire Debug Port

An optional external interface for the DAP that provides a serial-wire bidirectional debug interface.

Serial-Wire JTAG Debug Port

A standard debug port that combines JTAG-DP and SW-DP.

SW-DP

See Serial-Wire Debug Port.

SWJ-DP	<i>See</i> Serial-Wire JTAG Debug Port.
Synchronization primitive	The memory synchronization primitive instructions are those instructions that are used to ensure memory synchronization. That is, the LDREX and STREX instructions.
System memory	Memory space at 0x20000000 to 0xFFFFFFFF, excluding PPB space at 0xE0000000 to 0xE00FFFFF.
TAP	<i>See</i> Test access port.
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.
Thread Control Block	A data structure used by an operating system kernel to maintain information specific to a single thread of execution.
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.
TPA	<i>See</i> Trace Port Analyzer.
TPIU	<i>See</i> Trace Port Interface Unit.
Trace Port Interface Unit (TPIU)	Drains trace data and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.
Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
UNP	<i>See</i> 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.
Wake-up Interrupt Controller (WIC)	The Wake-up Interrupt Controller provides significantly reduced gate count interrupt detection and prioritization logic.

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. <i>See also</i> Breakpoint.
WIC	<i>See</i> Wake-up Interrupt Controller.
Word	A 32-bit data item.
Word-invariant	<p>In a word-invariant system, the address of each byte of memory changes when switching between little-endian and big-endian operation, in such a way that the byte with address A in one endianness has address A EOR 3 in the other endianness. As a result, each aligned word of memory always consists of the same four bytes of memory in the same order, regardless of endianness. The change of endianness occurs because of the change to the byte addresses, not because the bytes are rearranged.</p> <p>The ARM architecture supports word-invariant systems in ARMv3 and later versions. When word-invariant support is selected, the behavior of load or store instructions that are given unaligned addresses is instruction-specific, and is in general not the expected behavior for an unaligned access. It is recommended that word-invariant systems use the endianness that produces the required byte addresses at all times, apart possibly from very early in their reset handlers before they have set up the endianness, and that this early part of the reset handler must use only aligned word memory accesses.</p> <p><i>See also</i> Byte-invariant.</p>
Write buffer	A pipeline stage for buffering write data to prevent bus stalls from stalling the processor.