Arm® Cortex®-M85 Processor

Revision: r0p2

Technical Reference Manual

Non-Confidential

Copyright © 2020–2022 Arm Limited (or its affiliates).
All rights reserved.

Issue 05

101924_0002_05_en

Arm® Cortex®-M85 Processor

Technical Reference Manual

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document history

Issue	Date	Confidentiality	Change
0000-02	8 December 2020	Confidential	First beta release for r0p0
0000-03	30 June 2021	Confidential	First limited access release for r0p0
0001-04	15 November 2021	Confidential	First early access release for r0p1
0002-05	26 April 2022	Non-Confidential	First release for r0p2

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

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, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

1 Preface.....17
1.1 About this book.....17
1.1.1 Product revision status.....17
1.1.2 Intended audience.....17
1.1.3 Using this book.....17
1.1.4 Additional reading.....20
1.2 Feedback.....21
1.2.1 Feedback on this product.....21
1.2.2 Feedback on content.....21

2 Introduction.....22
2.1 Cortex®-M85 processor overview.....22
2.2 Cortex®-M85 features.....23
2.3 Supported standards and specifications.....25
2.4 Design tasks.....26
2.5 Documentation.....27
2.6 Product revisions.....28

3 Technical overview.....29
3.1 Cortex®-M85 processor components.....29
3.1.1 Cortex®-M85 processor core.....31
3.1.2 Extension Processing Unit.....32
3.1.3 Memory components.....33
3.1.4 Interrupt components.....35
3.1.5 Debug and trace components.....36
3.2 Interfaces.....37
3.3 Security.....38
3.4 Reliability.....39
3.5 Power intent.....39
3.6 Cortex®-M85 implementation options.....39

4 Programmers model.....42
4.1 Security states, operation, and execution modes.....42

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.2 Instruction set summary.. 43
4.3 Exclusive monitor.. 44
4.4 Cortex®-M85 processor core registers summary.. 44
4.5 Architectural registers.. 46
4.6 Exceptions.. 46
4.6.1 Exception handling and prioritization.. 47
4.6.2 Multicycle instructions.. 48

5 System registers.. 49
5.1 System control register summary.. 49
5.2 Identification register summary.. 52
5.2.1 Media and VFP Feature Register reset values, MVFR0, MVFR1, and MVFR2 reset values.... 56
5.3 AFSR, Auxiliary Fault Status Register.. 56
5.4 CPUID, CPUID Base Register.. 58
5.5 Cache identification register summary.. 59
5.5.1 CLIDR, Cache Level ID Register.. 60
5.5.2 CSSELR, Cache Size Selection Register.. 61
5.5.3 CCSIDR, Current Cache Size ID Register.. 62
5.6 REVIDR, Revision ID Register.. 64
5.7 Implementation control register summary.. 65
5.8 ACTLR, Auxiliary Control Register.. 65
5.9 ICTR, Interrupt Controller Type Register.. 67
5.10 IMPLEMENTATION DEFINED registers summary.. 68
5.11 Direct cache access registers.. 71
5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers.. 72
5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers.. 74
5.12 Error bank registers.. 78
5.12.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1.. 79
5.12.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1.. 80
5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1.. 82
5.13 MSCR, Memory System Control Register.. 84
5.14 PAHBCR, P-AHB Control Register.. 87
5.15 PFCR, Prefetcher Control Register.. 88
5.16 Power mode control registers.. 89
5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register.. 89
5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register.. 91

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.17 Processor configuration information registers....92
5.17.1 CFGINFOSEL, Processor configuration information selection register....92
5.17.2 CFGINFORD, Processor configuration information read data register.... 95
5.18 ID_PFR0, Processor Feature Register 0....96
5.19 ITCMCR and DTCMCR, TCM Control Registers.... 97
5.20 TCM security gate registers.... 98
5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers....99
5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers....100
5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers.... 101
5.21 EWIC interrupt status access registers.... 103
5.21.1 EVENTSPR, Event Set Pending Register.... 104
5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers.... 105

6 Initialization.... 107
6.1 Initialization overview....107
6.2 Initializing and reprogramming the MPU....107
6.3 Initializing the EPU.... 108
6.4 Programming the SAU.... 109
6.5 Initializing the instruction and data cache.... 109
6.5.1 Enabling the instruction and data cache.... 110
6.5.2 Powering down the caches....110
6.5.3 Powering up the caches.... 111
6.6 Enabling branch prediction.... 111
6.7 Enabling the branch cache.... 112
6.8 Enabling and preloading the TCM.... 112
6.9 Enabling and locking the TCM security gates....113
6.10 Enabling the P-AHB interface.... 114

7 Power management....115
7.1 Power domains.... 115
7.2 Power states....116
7.3 Power and operating mode transitions....117
7.3.1 Operating mode transitions which change PDRAMS power state.... 120
7.4 Core P-Channel and power mode selection.... 121
7.4.1 P-Channel interface tie-off when P-Channel is not used.... 122
7.5 COREPACTIVE and required power mode....123
7.5.1 COREPACTIVE signal encoding.... 125

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.6 PDCORE low-power requirements........125
7.7 PDEPU low-power requirements........126
7.8 PDRAMS powerdown requirements........127
7.9 Warm reset power mode........127
7.10 Debug Q-Channel and PDDEBUG power domain........129
7.11 Q-Channel clock control........130
7.12 PWRCOREWAKEPACTIVE........130
7.13 PWRDBGWAKEQACTIVE........131

8 Memory model........132
8.1 Memory map........132
8.2 Memory types........133
8.3 Private Peripheral Bus........134
8.4 Unaligned accesses........136
8.5 Access privilege level for Device and Normal memory........137
8.6 Memory ordering and barriers........138
8.7 Execute Only Memory........138

9 Memory Authentication........140
9.1 MAU features........140
9.2 Security Attribution Unit........140
9.2.1 SAU register summary........141
9.2.2 Security levels........141
9.3 Memory Protection Unit........142
9.3.1 Memory Protection Unit register summary........143
9.4 Implementation Defined Attribution Unit........144
9.4.1 IDAU interface and backwards compatibility........144
9.5 Memory regions not controlled by SAU and IDAU........145
9.6 Security attribution signals........145
9.7 TCM Gate Units........146
9.8 TCM and P-AHB security access control........147
9.8.1 Memory aliasing and IDAU/SAU configuration........147
9.8.2 Security access gating using the TGU........150
9.8.3 TGU configuration........150
9.8.4 Security check and fault response........152

10 Memory system........153

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.1 Memory system features........153
10.2 Memory system faults........155
10.2.1 Classes of fault........156
10.3 Memory system behavior........158
10.3.1 Speculative accesses........159
10.3.2 Access privilege level for Device and Normal memory........161
10.4 Master-AXI interface........162
10.4.1 High performance M-AXI configuration........162
10.4.2 Area optimized M-AXI configuration........165
10.4.3 Bridging to AHB........167
10.4.4 Write response........168
10.4.5 Memory system implications for AXI accesses........168
10.4.6 Master-AXI interface transfers........168
10.5 Peripheral AHB interface........169
10.5.1 P-AHB interface transfers........169
10.5.2 P-AHB interface configuration........171
10.5.3 P-AHB considerations........172
10.6 S-AHB interface........173
10.6.1 S-AHB memory map........174
10.6.2 S-AHB transfers........174
10.6.3 S-AHB interface arbitration........176
10.6.4 S-AHB availability and low power states........176
10.7 EPPB interfaces........177
10.8 TCM interfaces........178
10.8.1 TCM configuration........179
10.8.2 TCM transactions........180
10.8.3 Booting from TCM........181
10.8.4 Integration with flash memory........181
10.8.5 System access to TCM through the S-AHB DMA interface........182
10.9 Instruction and data cache........183
10.9.1 L1 data cache........184
10.9.2 L1 instruction cache........186
10.9.3 Cache maintenance operations........186
10.9.4 Automatic cache invalidation at reset........188
10.9.5 Cache coherency........188
10.9.6 Accessing the caches........189

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.9.7 System cache support......190
10.9.8 Direct cache access......190
10.10 Store buffer......193
10.10.1 Store buffer merging......193
10.10.2 Store buffer behavior......193
10.10.3 Store buffer ordering......194
10.10.4 Store buffer draining......194
10.11 Internal local exclusive access monitor......195
10.12 M-AXI and P-AHB interaction with the global exclusive monitor......196
10.13 MBIST......196

11 Reliability, Availability, and Serviceability Extension support......197
11.1 Cortex®-M85 processor implementation of RAS......197
11.1.1 Cortex®-M85 RAS events......198
11.2 ECC memory protection behavior......198
11.2.1 ECC schemes and error type terminology......199
11.2.2 Enabling ECC......200
11.2.3 Error detection and processing......200
11.2.4 Error reporting......204
11.2.5 Address decoder protection and white noise protection......206
11.3 RAS memory barriers......207
11.4 RAS Extension registers......207
11.4.1 ERRFR0, RAS Error Record Feature Register......208
11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register......209
11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers......212
11.4.4 ERRMISC10, Error Record Miscellaneous Register 10......213
11.4.5 ERRGSR0, RAS Fault Group Status Register......214
11.4.6 ERRDEVID, RAS Error Record Device ID Register......215
11.4.7 RFSR, RAS Fault Status Register......216

12 Nested Vectored Interrupt Controller......218
12.1 NVIC features......218
12.2 Registers associated with interrupt control and behavior......218
12.3 NVIC register summary......219
12.4 Software Interrupt Generation register summary......219
12.5 SysTick Timer register summary......220

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

13 External coprocessors......221
13.1 External coprocessors features......221
13.2 Operation......221
13.3 Data transfer rates......222
13.4 Coprocessor instruction restrictions......222
13.5 Debug access to coprocessor registers usage constraints......223
13.6 Exceptions and context switch......223
13.7 Response to coprocessor errors......223

14 Floating-point and MVE support......224
14.1 Floating-point and MVE operation......224
14.1.1 EPU views of the register bank......225
14.1.2 Modes of operation......225
14.1.3 Compliance with the IEEE 754 standard......225
14.1.4 Exceptions......225
14.2 Floating-point and MVE register summary......226
14.3 FPDSCR and FPSCR register reset values......226
14.4 Powering down the EPU......227

15 Debug......228
15.1 Debug functionality......228
15.1.1 CoreSight™ discovery......229
15.1.2 Debugger actions for identifying the processor......231
15.1.3 Processor ROM table identification and entries......231
15.1.4 Debug identification block register summary......233
15.1.5 Debug register summary......234
15.2 D-AHB interface......235
15.2.1 Debug memory access......235
15.2.2 Debugger access memory attributes and data cache access......237
15.2.3 Debug access security and attributes......239
15.2.4 Debug during reset and before code execution commences......240
15.2.5 Advanced DSP debug capabilities......241

16 Performance Monitoring Unit Extension......243
16.1 PMU features......243
16.2 PMU events......244
16.3 PMU register summary......249

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

17 Instrumentation Trace Macrocell....251
17.1 ITM features....251
17.2 ITM register summary....252
17.3 ITM_TPR, ITM Trace Privilege Register....254
17.4 ITM_ITCTRL, ITM Integration Mode Control Register....255
17.5 ITM_ITWRITE, Integration Write Register....255
17.6 ITM_ITREAD, Integration Read Register....256

18 Data Watchpoint and Trace unit....258
18.1 DWT features....258
18.2 DWT debug access control....259
18.3 DWT comparators....260
18.4 Cycle counter and profiling counters....262
18.5 DWT register summary....263

19 Cross Trigger Interface....266
19.1 CTI features....266
19.2 CTI register summary....268
19.3 CTI_CONTROL, CTI Control Register....269
19.4 CTI_INTACK, CTI Interrupt Acknowledge Register....270
19.5 CTI_APPSET, CTI Application Channel Set Register....271
19.6 CTI_APPCLR, CTI Application Channel Clear Register....272
19.7 CTI_APPPULSE, CTI Application Channel Pulse Register....273
19.8 CTI_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register....274
19.9 CTI_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register....275
19.10 CTI_TRIGINSTATUS, CTI Trigger Input Status Register....276
19.11 CTI_TRIGOUTSTATUS, CTI Trigger Output Status Register....277
19.12 CTI_CHINSTATUS, CTI Channel Input Status Register....278
19.13 CTI_CHOUTSTATUS, CTI Channel Output Status Register....279
19.14 CTI_CHANNELGATE, CTI Channel Gate Register....280
19.15 CTI_ITCHOUT, Integration Test Channel Output Register....281
19.16 CTI_ITTRIGOUT, Integration Test Trigger Output Register....282
19.17 CTI_ITCHIN, Integration Test Channel Input Register....283
19.18 CTI_ITTRIGIN, Integration Test Trigger Input Register....284
19.19 CTI_ITCONTROL, Integration Mode Control Register....285
19.20 CTI_DEVARCH, Device Architecture Register....285
19.21 CTI_DEVID, Device Configuration Register....286

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.22 CTI_DEVTYPE, Device Type Identifier Register....287
19.23 CTI_PIDR4, Peripheral Identification Register 4....288
19.24 CTI_PIDR5, Peripheral Identification Register 5....289
19.25 CTI_PIDR6, Peripheral Identification Register 6....289
19.26 CTI_PIDR7, Peripheral Identification Register 7....290
19.27 CTI_PIDR0, Peripheral Identification Register 0....291
19.28 CTI_PIDR1, Peripheral Identification Register 1....292
19.29 CTI_PIDR2, Peripheral Identification Register 2....293
19.30 CTI_PIDR3, Peripheral Identification Register 3....294
19.31 CTI_ CIDR0, Component Identification Register 0....294
19.32 CTI_ CIDR1, Component Identification Register 1....295
19.33 CTI_ CIDR2, Component Identification Register 2....296
19.34 CTI_ CIDR3, Component Identification Register 3....297

20 BreakPoint Unit....298
20.1 BPU features....298
20.2 BPU register summary....298

A External Wakeup Interrupt Controller....300
A.1 EWIC features....300
A.2 EWIC register summary....301
A.2.1 EWIC_CR, EWIC Control Register....301
A.2.2 EWIC_ASCR, EWIC Automatic Sequence Control Register....302
A.2.3 EWIC_CLRMASK, EWIC Clear Mask Register....304
A.2.4 EWIC_NUMID, EWIC Event Number ID Register....304
A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers....305
A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers....306
A.2.7 EWIC_PSR, EWIC Pend Summary Register....308
A.2.8 EWIC CoreSight™ register summary....309
A.2.9 EWIC_CLAIMSET, EWIC Claim Tag Set Register....310
A.2.10 EWIC_CLAIMCLR, EWIC Claim Tag Clear Register....311

B Trace Port Interface Unit....313
B.1 TPIU features....313
B.1.1 TPIU Formatter....314
B.1.2 Serial Wire Output format....315
B.2 TPIU register summary....315

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.1 TPIU_SSPSR, Supported Port Size Register.......317
B.2.2 TPIU_CSPSR, Current Port Size Register.......318
B.2.3 TPIU_SPPR, Selected Pin Protocol Register.......319
B.2.4 TPIU_PSCR, Periodic Synchronization Counter Register.......320
B.2.5 TPIU_ACPR, Asynchronous Clock Prescaler Register.......320
B.2.6 TPIU_FFSR, Formatter and Flush Status Register.......321
B.2.7 TPIU_FFCR, Formatter and Flush Control Register.......322
B.2.8 TPIU_TRIGGER, TPIU TRIGGER Register.......323
B.2.9 ITFTTD0, Integration Test FIFO Test Data 0 Register.......324
B.2.10 ITATBCTR2, Integration Test ATB Control Register 2.......325
B.2.11 ITFTTD1, Integration Test FIFO Test Data 1 Register.......325
B.2.12 ITATBCTR0, Integration Test ATB Control 0 Register.......326
B.2.13 TPIU_ITCTRL, Integration Mode Control.......327
B.2.14 CLAIMSET, Claim Tag Set Register.......328
B.2.15 CLAIMCLR, Claim Tag Clear Register.......329
B.2.16 TPIU_DEVID, Device Configuration Register.......330
B.2.17 TPIU_DEVTYPE, Device Type Identifier Register.......331
B.2.18 TPIU_PIDR4, Peripheral Identification Register 4.......331
B.2.19 TPIU_PIDR5, Peripheral Identification Register 5.......332
B.2.20 TPIU_PIDR6, Peripheral Identification Register 6.......333
B.2.21 TPIU_PIDR7, Peripheral Identification Register 7.......334
B.2.22 TPIU_PIDR0, Peripheral Identification Register 0.......334
B.2.23 TPIU_PIDR1, Peripheral Identification Register 1.......335
B.2.24 TPIU_PIDR2, Peripheral Identification Register 2.......336
B.2.25 TPIU_PIDR3, Peripheral Identification Register 3.......337
B.2.26 TPIU_ CIDR0, Component Identification Register 0.......338
B.2.27 TPIU_ CIDR1, Component Identification Register 1.......338
B.2.28 TPIU_ CIDR2, Component Identification Register 2.......339
B.2.29 TPIU_ CIDR3, Component Identification Register 3.......340

C Signal descriptions.......341
C.1 Clock and clock enable signals.......341
C.2 Reset signals.......341
C.3 Static configuration signals.......342
C.4 Reset configuration signals.......344
C.5 Cache initialization signal.......345

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

C.6 Instruction execution control signals 345
C.7 Instruction Tightly Coupled Memory interface signals 346
C.8 Data Tightly Coupled Memory interface signals 348
C.9 M-AXI interface signals 349
C.9.1 M-AXI interface protection signals 352
C.10 S-AHB interface signals 353
C.10.1 S-AHB interface protection signals 354
C.11 P-AHB interface signals 354
C.11.1 P-AHB interface protection signals 355
C.12 D-AHB interface signals 356
C.12.1 D-AHB interface protection signals 357
C.13 Debug EPPB interface signals 357
C.13.1 Debug EPPB interface protection signals 358
C.14 Core EPPB interface signals 358
C.14.1 Core EPPB interface protection signals 359
C.15 External coprocessor interface signals 360
C.16 Debug interface signals 360
C.17 P-Channel and Q-Channel power control signals 361
C.18 Q-Channel clock control signals 362
C.19 Power compatibility control signals 363
C.20 ITM interface signals 363
C.21 ETM interface signals 364
C.22 Trace synchronization and trigger signals 364
C.23 CTI interface signals 364
C.24 Interrupt signals 365
C.25 WIC interface signals 365
C.26 Event signals 367
C.27 IDAU interface signals 367
C.28 Miscellaneous signals 368
C.29 Error interface signals 372
C.30 Floating-point exception signals 373
C.31 PMC-100 interface signals 374
C.32 Test interface signals 375
C.33 DCLS operation signals 376

D UNPREDICTABLE Behaviors 377

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

D.1 Use of instructions defined in architecture variants....... 377
D.2 Use of Program Counter - R15 encoding....... 377
D.3 Use of Stack Pointer - as a general-purpose register R13....... 377
D.4 Register list in load and store multiple instructions....... 378
D.5 Exception-continuable instructions....... 378
D.6 Stack limit checking....... 379
D.7 UNPREDICTABLE instructions within an IT block....... 379
D.8 Memory access and address space....... 380
D.9 MPU programming....... 381
D.10 Miscellaneous UNPREDICTABLE instruction behavior....... 382

E Revisions....... 383
E.1 Revisions....... 383

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1 Preface

1.1 About this book

Information about the revision status and the intended audience for the book.

1.1.1 Product revision status

The r*x*p*y* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*x* Identifies the major revision of the product, for example, r1.

p*y* Identifies the minor revision or modification status of the product, for example, p2.

1.1.2 Intended audience

This manual is written to help system designers, system integrators, verification engineers, and software programmers who are implementing a *System on Chip* (SoC) device based on the Cortex®-M85 processor.

Using this book

Information about the chapters in this book.

This book is organized into the following chapters:

1 Introduction

2 Technical overview

3 Programmers model

4 System registers

5 Initialization

6 Power management

7 Memory model

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8 Memory Authentication

9 Memory system

10 Reliability, Availability, and Serviceability Extension support

11 Nested Vectored Interrupt Controller

12 External coprocessors

13 Floating-point and MVE support

14 Debug

15 Performance Monitoring Unit Extension

16 Instrumentation Trace Macrocell

17 Data Watchpoint and Trace unit

18 Cross Trigger Interface

19 BreakPoint Unit

Glossary

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

See the Arm Glossary for more information.

Typographic conventions

The typographic conventions used throughout this document are as follows.

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace italic

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

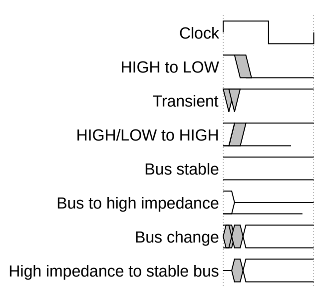
Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The following figure 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.

Figure 1-1: Key to timing diagram conventions



Signals

The signal conventions are:

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name, n denotes an active-LOW signal.

1.1.4 Additional reading

Information published by Arm and by third parties.

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

Arm publications

- *Arm®v8-M Architecture Reference Manual* (DDI 0553)
- *Arm® AMBA® 5 AHB Protocol Specification* (IHI 0033)
- *AMBA® APB Protocol Version 2.0 Specification* (IHI 0033)
- *AMBA® ATB Protocol Specification* (IHI 0032)
- *AMBA® AXI and ACE Protocol Specification* (IHI 0022)
- *Arm® CoreSight™ Architecture Specification v3.0* (IHI 0029)
- *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* (100806)
- *Arm® CoreSight™ TPIU-M Technical Reference Manual* (102427)
- *Arm® Debug Interface Architecture Specification, ADIv6.0* (IHI 0074)
- *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* (ARM IHI 0064)
- *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces* (IHI 0068)
- *Arm® PMC-100 Technical Reference Manual* (101528)
- *Arm® Reliability, Availability, and Serviceability (RAS) Specification* (DDI 0587)
- *Arm® CoreSight™ ETM-M85 Technical Reference Manual* (101926)

The following confidential book is only available to licensees:

- *Arm® Cortex®-M85 Processor Integration and Implementation Manual* (101925)

Other publications

- *IEEE Std 1149.1-2001, Test Access Port and Boundary-Scan Architecture (JTAG)*
- *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.2 Feedback

Arm welcomes feedback on this product and its documentation.

1.2.1 Feedback on this product

Information about how to give feedback on the product.

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

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

1.2.2 Feedback on content

Information about how to give feedback on the content.

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

- The title Arm® Cortex®-M85 Processor Technical Reference Manual.
- The number 101924_0002_05_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2 Introduction

This chapter provides an overview of the Cortex®-M85 processor and its features.

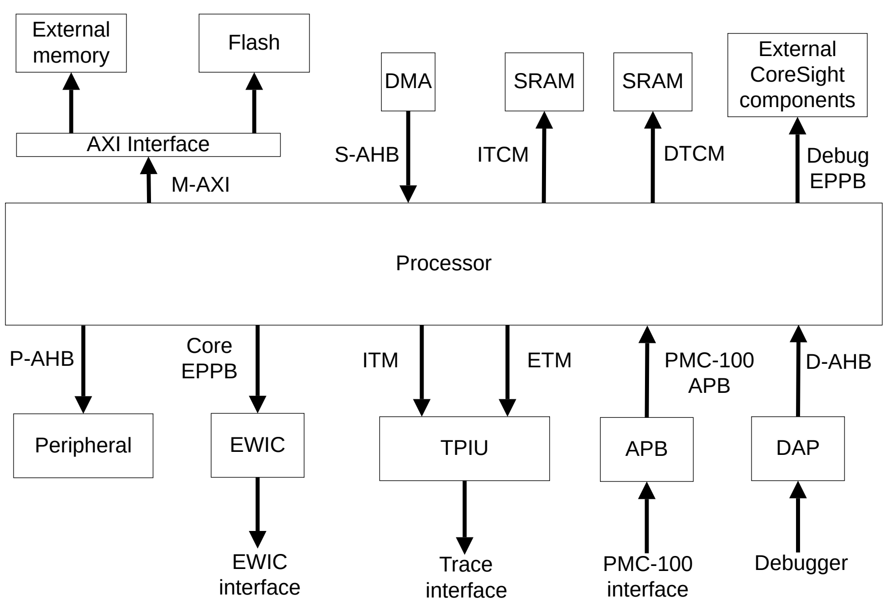
2.1 Cortex®-M85 processor overview

The Cortex®-M85 processor is a fully synthesizable high-performance microcontroller class processor that implements the Arm®v8.1-M Mainline architecture which includes support for the *M-profile Vector Extension* (MVE). The processor also supports previous Arm®v8-M architectural features.

The design is focused on compute applications such as *Digital Signal Processing* (DSP) and machine learning. The Cortex®-M85 processor is energy efficient and achieves high compute performance across scalar and vector operations while maintaining low power consumption.

The following figure shows the Cortex®-M85 processor in a typical system.

Figure 2-1: Example processor system



Note

The *Programmable MBIST Controller* (PMC-100) will be supported in a future processor release, as part of the Cortex®-M85 safety package. The PMC-100 APB interface is reserved.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For more information on the processor-level components, see 3.1 Cortex-M85 processor components on page 29.

2.2 Cortex®-M85 features

The Cortex®-M85 processor implements the Arm®v8.1-M Mainline architecture and also supports previous Arm®v8-M architectural features.

For more information on Arm®v8-M and Arm®v8.1-M features and variants information, see the *Arm®v8-M variants* section in the *Arm®v8-M Architecture Reference Manual*.

Note

- The 'Optional' column indicates a feature that can be optionally included, either by:
  - Setting relevant *Register Transfer Level* (RTL) parameters. For example, if you include the *Instrumentation Trace Macrocell* (ITM).
  - Being optionally licensed. For example, if you optionally license ETM-Cortex®-M85.
- The 'Configurable' column indicates a feature that can be configured to any permitted value by setting relevant RTL parameters. For example, you can configure the size of the instruction and data cache to be 4KB, 8KB, 16KB, 32KB, or 64KB.

Table 2-1: Cortex®-M85 processor architectural features

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
Arm PMSAv8 memory system architecture with memory protection	-	Yes	-	-	-
Arm FPv5 hardware that supports scalar half-, single-, and double-precision floating-point operation and is compliant with IEEE754-2008	Arm®v8-M onwards	-	Yes	-	Optionally licensable component
Digital Signal Processing (DSP) Extension	Arm®v8-M onwards	Yes	-	-	-
Digital Signal Processing (DSP) Debug Extension	Arm®v8.1-M	Yes	-	-	-
Exception model	Arm®v8-M onwards	Yes	-	-	See 4.6 Exceptions on page 46 for more information
External *Implementation Defined Attribution Unit* (IDAU)	-	Yes	-	-	-
Level 1 (L1) instruction and data cache	Arm®v8-M onwards	-	Yes	Yes	-
Main Extension	Arm®v8.1-M	Yes	-	-	Includes the 16-bit and 32-bit Thumb instruction set
Memory Protection Unit (MPU)	Arm®v8-M onwards	-	Yes	Yes	Supports up to 16 regions each for Secure and Non-secure applications

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
M-profile Vector Extension (MVE), supporting *Single Instruction Multiple Data* (SIMD) 128-bit vector operations	Arm®v8.1-M	-	Yes	-	Supported data types: • Integer • Half precision floating-point (supported when floating-point functionality is included) • Single precision floating-point (supported when floating-point functionality is included) MVE is also referred to as Arm® Helium™ technology.
Support for *Data Independent Timing* (DIT) operation	Arm®v8.1-M	Yes	-	-	See the *Arm®v8-M Architecture Reference Manual*
Nested Vector Interrupt Controller (NVIC)	Arm®v8-M onwards	Yes	-	Yes	Supports up to 480 external interrupts with up to 256 priority levels
Reliability, Availability, and Serviceability (RAS) Extension	Arm®v8.1-M	Yes	-	-	-
Security Attribution Unit (SAU)	Arm®v8-M onwards	-	Yes	Yes	Supports up to eight Non-secure or Non-secure Callable memory regions
Security Extension	Arm®v8-M onwards	Yes	-	-	The Security Extension is an implementation of Arm® TrustZone® technology.
Unprivileged Debug Extension (UDE)	Arm®v8.1-M	Yes	-	-	-
Pointer Authentication and Branch Target Identification (PACBTI) Extension	Arm®v8.1-M	-	Yes	-	For more information about the PACBTI Extension, see the *Arm®v8-M Architecture Reference Manual*.

Debug and trace features

The following table shows the debug and trace features of the processor.

Table 2-2: Debug and trace features

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
BreakPoint Unit (BPU)	Arm®v8-M onwards	Yes	-	Yes	Supports up to eight comparators
Cross Trigger Interface (CTI) unit	Arm®v8-M onwards	Yes	-	-	-
Data Watchpoint and Trace (DWT) unit	Arm®v8-M onwards	Yes	-	Yes	Supports up to eight comparators
Embedded Trace Macrocell (ETM)	Arm (ETM) v4.5	-	Yes	-	Optionally licensable component
Instrumentation Trace Macrocell (ITM)	Arm®v8-M onwards	-	Yes	-	-

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
Performance Monitoring Unit (PMU)	Arm®v8.1-M	Yes	-	-	-

2.3 Supported standards and specifications

The Cortex®-M85 processor complies with, or implements, the relevant Arm architectural standards and protocols.

This book complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards. It does not duplicate information from these sources.

Arm architecture

The Cortex®-M85 processor is compliant with the Arm®v8.1-M Mainline architecture and also supports previous Arm®v8-M architectural features. See 2.2 Cortex-M85 features on page 23 for more information.

Bus architecture

The Cortex®-M85 processor implements AMBA® 5 AXI-compliant *Master AXI* (M-AXI) interface for slow on-chip or off-chip memory and devices.

It also provides external interfaces that comply with the AMBA® 5 AHB protocol.

Additionally, the Cortex®-M85 processor implements interfaces for CoreSight™ and other debug components using the AMBA® 4 APB protocol (this is the same as APB protocol version 2.0) and the ATBv1.1 part of the AMBA® 4 ATB protocol.

For more information, see the:

- *AMBA® AXI and ACE Protocol Specification*
- *Arm® AMBA® 5 AHB Protocol Specification*
- *AMBA® APB Protocol Version 2.0 Specification*
- *AMBA® ATB Protocol Specification*

The Cortex®-M85 processor also provides P-Channel and Q-Channel interfaces for power and clock control. See the *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.

For more overview information on bus interfaces, see 3.2 Interfaces on page 37.

Debug

The debug features of the Cortex®-M85 processor implement the Arm Debug Interface v6.0 architecture.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

See the *Arm® Debug Interface Architecture Specification, ADIv6.0*.

Embedded Trace Macrocell

The trace features of the Cortex®-M85 processor implement the Arm *Embedded Trace Macrocell* (ETM) v4.5 architecture.

See the *Arm® CoreSight™ ETM-M85 Technical Reference Manual* for more information on ETM-Cortex®-M85 which is an optional component that you can license.

Extension Processing Unit

The *Extension Processing Unit* (EPU) performs scalar floating-point and vector operations.

The EPU is configured to include a scalar floating-point functionality, which supports half-precision, single-precision, and double-precision arithmetic as defined by the Arm FPv5 architecture.

The EPU implements MVE, which can support:

- Half-precision, single-precision, and double-precision floating-point
- Integer, half-precision, and single-precision vector arithmetic

See 3.6 Cortex-M85 implementation options on page 39.

The Cortex®-M85 processor provides floating-point computation functionality that is included with Floating-point and MVE, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*.

2.4 Design tasks

The Cortex®-M85 processor is delivered as synthesizable RTL that must go through implementation, integration, and programming processes before you can use it in a product.

The following definitions describe each top-level process in the design flow:

Implementation
The implementer configures and synthesizes the RTL.

Integration
The integrator connects the Cortex®-M85 processor into an SoC. This includes connecting it to a memory system and peripherals.

Programming
The system programmer develops the software required to configure and initialize the Cortex®-M85 processor and tests the required application software.

Implementation and integration choices affect the behavior and features of the Cortex®-M85 processor.

The operation of the final device depends on:

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and features of the resulting macrocell.

Configuration inputs

The integrator configures some features of the Cortex®-M85 processor by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made. They can also limit the options available to the software.

Software configuration

The programmer configures the Cortex®-M85 processor by programming particular values into registers. This affects the behavior of the Cortex®-M85 processor.

Note

This manual refers to **IMPLEMENTATION-DEFINED** features that are applicable to build configuration options. Reference to a feature that is included means that the appropriate build and signal configuration options have been selected. Reference to an enabled feature means that software has also configured the feature.

2.5 Documentation

The Cortex®-M85 processor documentation can help you complete the top-level processes of implementation, integration, and programming that are required to use the product correctly.

The Cortex®-M85 processor documentation includes a Technical Reference Manual, an Integration and Implementation Manual, and User Guide Reference Material.

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex®-M85 processor. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the Cortex®-M85 processor is implemented and integrated. If you are programming the Cortex®-M85 processor, then contact the implementer to determine:

- The build configuration of the implementation.
- What integration, if any, was performed before implementing the Cortex®-M85 processor.

Integration and Implementation Manual

The *Integration and Implementation Manual* (IIM) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the Cortex®-M85 processor into an SoC. This includes a description of the integration kit and describes the pins that the integrator must tie off to configure the macrocell for the required integration.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- How to implement the Cortex®-M85 processor into your design. This includes *Memory Built-In Self Test* (MBIST) and *Design for Test* (DFT) information, and information on how to perform netlist dynamic verification on the Cortex®-M85 processor.
- The processes to sign off the integration and implementation of the design.

The Arm product deliverables include reference scripts and information about using them to implement your design.

Reference methodology documentation from your EDA tools vendor and the *implementation Reference Methodology* (iRM) `readme.txt` provided by Arm complements the IIM.

The IIM is a confidential book that is only available to licensees and Arm partners with an NDA agreement.

User Guide Reference Material

This document provides reference material that Arm partners can configure and include in a User Guide for an Arm Cortex®-M85 processor. Typically:

- Each chapter in this reference material might correspond to a section in the User Guide.
- Each top-level section in this reference material might correspond to a chapter in the User Guide.

However, you can organize this material in any way, subject to the conditions of the license agreement under which Arm supplied the material.

See the 1.1.4 Additional reading on page 20 for more information about the books that are associated with the Cortex®-M85 processor.

2.6 Product revisions

The following product revisions have been released.

r0p0 First beta release for r0p0
r0p1 First early access release for r0p1

r0p2

First release for r0p2

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

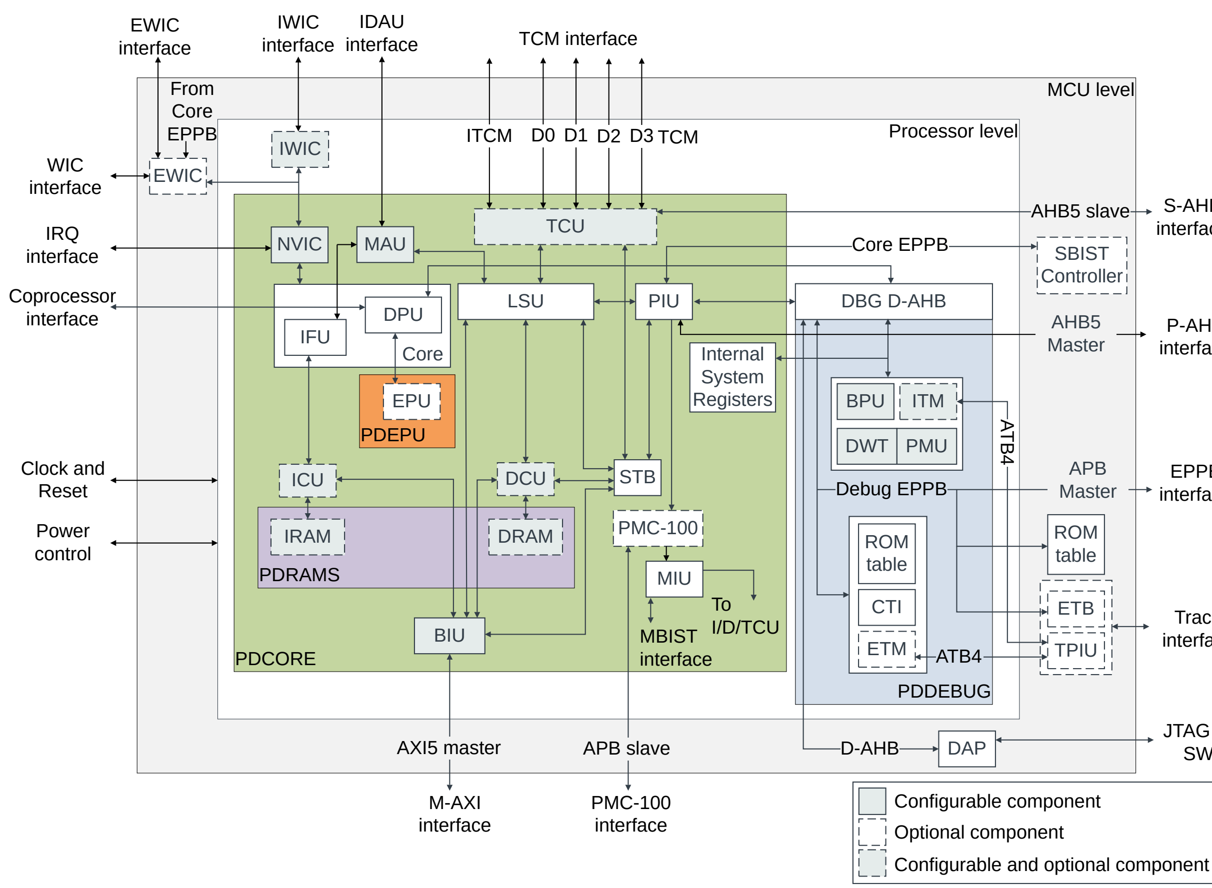
3 Technical overview

This chapter describes the Cortex®-M85 processor components and configuration options.

3.1 Cortex®-M85 processor components

The Cortex®-M85 processor has fixed and optional component blocks.

Figure 3-1: Cortex®-M85 processor block diagram



> **Note**
>
> - The *Programmable MBIST Controller* (PMC-100) and *Software Built-In Self-Test* (SBIST) Controller components are reseved for future use in the Cortex®-M85 processor .
> - For more information on the PDCORE, PDDEBUG, PDEPU, and PDRAMS power domains, and their clocking, reset, and power requirements, see 7 Power management on page 115.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the various processor components shown in the processor block diagram.

Table 3-1: Processor components

Block	Component
Processor core	The Cortex®-M85 processor core has an *Instruction Fetch Unit* (IFU) that is closely coupled with the *Data Processing Unit* (DPU). For more information, see 3.1.1 Cortex-M85 processor core on page 31.
Extension Processing Unit (EPU)	The EPU performs: • Scalar floating-point operations • *M-class Vector Extension* (MVE) operations For more information, see 3.1.2 Extension Processing Unit on page 32. The EPU can be optionally included.

Memory components	The memory components are:
	• *Memory Authentication Unit* (MAU). For more information on the MAU, see 3.1.3.1 Memory Authentication Unit on page 33 . The MAU contains:
	  ◦ *Security Attribution Unit* (SAU)
	  ◦ *TCM Gate Unit* (TGU)
	  ◦ Secure MPU region, MPU_S, which is always optionally configured
	  ◦ Non-secure MPU region, MPU_NS, which is always optionally configured
	• *Load Store Unit* (LSU)
	• *Peripheral Interface Unit* (PIU)
	• *TCM Control Unit* (TCU)
	• *Data Cache Unit* (DCU) and *Data RAM* (DRAM)
	• *Instruction Cache Unit* (ICU) and *Instruction RAM* (IRAM)
	• *Bus Interface Unit* (BIU)
	• *STore Buffer* (STB)
	• *MBIST Interface Unit* (MIU)
	
	For more information on the memory system, see 3.1.3.2 Memory system on page 34.

Interrupt components	The interrupt components are: • *Nested Vectored Interrupt Controller* (NVIC) • *External Wakeup Interrupt Controller* (EWIC), which can be optionally included • *Internal Wakeup Interrupt Controller* (IWIC), which can be optionally included For more information on the interrupt-related components, see 3.1.4 Interrupt components on page 35.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Block	Component
Debug and trace components	The debug and trace components are: • *BreakPoint unit* (BPU) • *Cross Trigger Interface* (CTI) • CoreSight™-compliant *Debug Access Port* (DAP), CoreSight™ DAP-Lite2, which is available for download when you license Cortex®-M85 processor IP. • *Data Watchpoint and Trace* (DWT) unit • *Performance Monitoring Unit* (PMU), which is located in the DWT • *Embedded Trace Macrocell* (ETM), which is an optional licensable component. • *Instrumentation Trace Macrocell* (ITM) • *Trace Port Interface Unit* (TPIU) • CoreSight™-compliant *Embedded Trace Buffer* (ETB) functionality support. The ETB is not delivered as a part of the IP deliverable. The ETB is an optional licensable component which is available when you license either the CoreSight™ SoC-600 or CoreSight™ SoC-600M. The Cortex®-M85 IP deliverable has a placeholder for ETB integration. For more information on the debug and trace related components, see 3.1.5 Debug and trace components on page 36.

3.1.1 Cortex®-M85 processor core

The Cortex®-M85 processor core has an *Instruction Fetch Unit* (IFU) that is closely coupled with the *Data Processing Unit* (DPU).

The DPU contains the logic to:

- Decode and execute scalar integer instructions
- Handle the register transfer operations required for exception entry and exit

The Cortex®-M85 processor core has the following features:

- An in-order pipeline with:
    - A 7-stage scalar pipeline
    - A 9-10 stage vector and floating point pipeline
- Multiple arithmetic logic shift units that support:
    - Regular shift and arithmetic operations
    - The SIMD operations included in the *Digital Signal Processing* (DSP) Extension, including full dual issue for most regular operations
    - The ability to accept late data bypass for most operations
- The core can handle up to two 32-bit vector load operations in parallel, when *M-profile Vector Extension* (MVE) is configured in the Cortex®-M85 processor.
- Harvard bus interfaces with vector fetch capability on the instruction side to optimize exception entry for efficient operation of compute workloads:
    - 64-bit instruction fetch data width

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - 64-bit load/store data width
- Optimized set of integer register bank ports for energy-efficient operation
- Integer divide unit with support for operand-dependent early termination
    - In this context, early termination refers to operations that terminate sooner than the expected number of cycles for the integer divide unit. Early termination capabilities depend on the data that enters the pipeline.
- Single cycle branch latency in most instances, with enhanced branch prediction capability
- Dual-issue of most 16-bit and 32-bit instructions
- Fully pipelined load and store operations for minimal stalls of memory access to data processing operations
- Dual issue of load and store operations
- Support for exception-continuable load and store multiple accesses
- Instruction queue to decouple instruction fetching and instruction execution
- Data prefetch to minimize the effect of AXI latency when accessing consistent patterns of cacheable data.

Note

The Cortex®-M85 processor core works with the *Extension Processing Unit* (EPU), when configured to provide full support for:

- Integer and floating-point operations included in MVE
- Scalar half-precision, single-precision, and double-precision floating-point operations

---

3.1.2 Extension Processing Unit

The *Extension Processing Unit* (EPU) includes support for all the instructions in the *M-profile Vector Extension* (MVE) and half, single, and double-precision scalar FPv5 architecture.

The EPU has the following features:

- MVE is implemented using a 64-bit arithmetic and load/store data-path in a two beats per tick configuration. A beat is the execution of ¼ of an MVE instruction. Instructions can overlap to allow full utilization of the logic with a sustained bandwidth of 64-bit *Multiply ACcumulate* (MAC) and 64-bit load/store per cycle. For more information on vector operation terminology, see *Arm®v8-M Architecture Reference Manual*.
- Extended register file, which is optimized for efficient vector operations.
- Floating-point MAC unit capable of a throughput of up to two single-precision or four-half precision MAC instructions every cycle when MVE is included in the Cortex®-M85 processor, or one single or half-precision MAC every cycle when only scalar floating-point is configured.
- Area optimized double-precision floating-point implementation.
- Support for Security Extension including lazy context stacking.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.1.3 Memory components

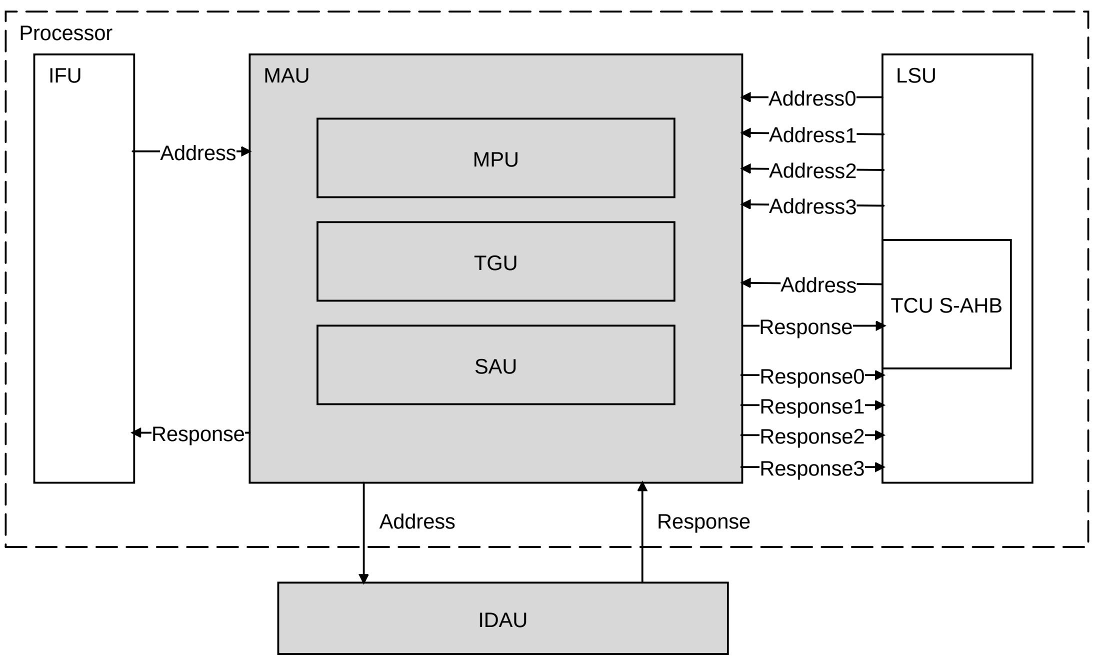
The Cortex®-M85 processor memory components consist of the *Memory Authentication Unit* (MAU) and memory system interfaces.

3.1.3.1 Memory Authentication Unit

The Cortex®-M85 processor *Memory Authentication Unit* (MAU) contains several units that control access to the memory.

The following figure shows the MAU block diagram.

Figure 3-2: MAU block diagram



Memory Protection Unit

The *Memory Protection Unit* (MPU) supports the Arm *Protected Memory System Architecture* (PMSA). Therefore, the MPU provides programmable support for memory protection using many software controllable regions. This unit defines the memory attributes that are associated with a particular memory region and the access permissions of addresses. Memory regions can be programmed to generate faults when accessed inappropriately, for example, by unprivileged software, reducing the scope of incorrectly written application code. The

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

architecture includes fault status registers to allow an exception handler to determine the source of the fault and to apply corrective action or notify the system.

The entire MPU logic can be split into Secure and Non-secure MPU regions.

Security Attribution Unit

The *Security Attribution Unit* (SAU) defines and authenticates accesses to memory based on the Security state of the core or the debugger. These states can be any of the following:

- Non-secure.
- Secure and Non-secure Callable.
- Secure.

TCM Gate Unit

The *TCM Gate Unit* (TGU) controls software and *Slave AHB* (S-AHB) accesses to the TCMs based on the security attribute of the access.

Interface to the IDAU

The MAU contains an interface to the *Implementation Defined Attribution Unit* (IDAU), which is present outside the core and not a part of the Cortex®-M85 processor. This unit defines memory regions as being either Secure, Non-secure, Non-secure Callable, or exempt from security checking. The final security mapping of memory regions is a combination of the response from the SAU and IDAU.

3.1.3.2 Memory system

The Cortex®-M85 processor memory system provides the interface between the core and the caches, external memory interfaces, and internal memory-mapped registers.

The memory system includes:

- A single interface to an *Instruction Tightly Coupled Memory* (ITCM) and four interfaces to *Data Tightly Coupled Memories* (DTCMs), D0TCM, D1TCM, D2TCM, and D3TCM
- A *Master AXI* (M-AXI) interface that can be used for on-chip or off-chip memory and devices
- A *Peripheral AHB* (P-AHB) interface for access to external peripherals
- A *Slave AHB* (S-AHB) interface for system access to the TCMs
- An L1 instruction cache
- An L1 data cache
- Two *External Private Peripheral Bus* (EPPB) interfaces:
  - Debug EPPB, for connecting to CoreSight™ debug and trace components, TPIU, ETB, MCU Rom Table and other external debug peripherals
  - Core EPPB, for connecting to EWIC
- A *STore Buffer* (STB) to hold store operations when they have left the load/store pipeline and the DPU has committed them. From the STB, a store can do any of the following:
  - Request access to the cache RAM through the DCU

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- ◦ Request the *Bus Interface Unit* (BIU) to initiate linefills
- ◦ Request the BIU to write data on the M-AXI interface
- ◦ Request access to the instruction or data TCM through the TCU
- ◦ Request access to the internal peripheral registers associated with the *Private Peripheral Bus* (PPB)

If there are several store transactions that are associated with the same 64-bit aligned doubleword, the STB can merge these store transactions into a single transaction.

For more information, see:

- 8 Memory model on page 132.
- 3.1.3.2 Memory system on page 34.

3.1.4 Interrupt components

The Cortex®-M85 processor interrupt components are responsible for low-latency interrupt processing and enabling the Cortex®-M85 processor to enter and wake up from low-power state.

3.1.4.1 NVIC features

The Cortex®-M85 processor *Nested Vectored Interrupt Controller* (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

The NVIC is responsible for:

- Maintaining the current execution priority of the Cortex®-M85 processor.
- Maintaining the pending and active status of all exceptions that are supported.
- Invoking preemption when a pending exception has priority.
- Providing wakeup signals to wakeup the Cortex®-M85 processor from deep sleep mode.
- Providing support to the *Internal Wakeup Interrupt Controller* (IWIC) and *External Wakeup Interrupt Controller* (EWIC).
- Providing priority and exception information to other processor components.

The NVIC in the Cortex®-M85 processor allows up to 496 exceptions, of which, 480 can be regular external interrupts.

3.1.4.2 Wakeup Interrupt Controller

The Cortex®-M85 processor supports a *Wakeup Interrupt Controller* (WIC) unit that allows the Cortex®-M85 processor to enter low-power state.

There are two WICs that are supported:

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- An *Internal Wakeup Interrupt Controller* (IWIC) that is synchronous with the processor and contained within the Cortex®-M85 processor boundary.
- An *External Wakeup Interrupt Controller* (EWIC), which is a system-level component that can be asynchronous to the Cortex®-M85 processor.

The Cortex®-M85 processor supports any of the following:

- No WIC.
- IWIC only.
- EWIC only.
- Both IWIC and EWIC.

3.1.5 Debug and trace components

The Cortex®-M85 processor supports multiple debug and trace components.

BreakPoint Unit

A configurable *BreakPoint Unit* (BPU) for implementing breakpoints.

Data Watchpoint and Trace

A configurable *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.

Instrumentation Trace Macrocell

An optional *Instrumentation Trace Macrocell* (ITM) that supports `printf()` style debugging using instrumentation trace.

Performance Monitoring Unit

A *Performance Monitoring Unit* (PMU) which enables software and debugger to gather statistics on events taking place on the Cortex®-M85 processor. These statistics can be used for performance analysis and system debug.

The PMU is always included.

ROM tables

ROM tables allow debuggers to determine which CoreSight™ components are implemented in the Cortex®-M85 processor.

Debug and trace interfaces

These interfaces are suitable for:

- Passing on-chip data through a *Trace Port Interface Unit* (TPIU) to a *Trace Port Analyzer* (TPA), including *Serial Wire Output* (SWO) mode
- Integrating a *Debug Access Port* (DAP), which is a debug port that is used to control debug functionality
- Integrating a CoreSight™ *Embedded Trace Buffer* (ETB), which is an optional licensable component for trace data to be written to an external SRAM

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Cross Trigger Interface

The *Cross Trigger Interface* (CTI) enables the debug logic and *Embedded Trace Macrocell* (ETM) to interact with each other and with other CoreSight™ components.

Embedded Trace Macrocell

The optional **EMBEDDED TRACE MACROCELL** (ETM) provides instruction-only trace capabilities. For more information, see the *Arm® CoreSight™ ETM-M85 Technical Reference Manual*.

3.2 Interfaces

The following table summarizes the interfaces that the Cortex®-M85 processor supports.

Note

For more information on the protocols in the following table, refer to the following specifications:

- *Arm® AMBA® 5 AHB Protocol Specification*
- *AMBA® APB Protocol Version 2.0 Specification*
- *AMBA® ATB Protocol Specification*
- *AMBA® AXI and ACE Protocol Specification*

Table 3-2: Interfaces

Name	Protocol	Width	Details
Master AXI (M-AXI) interface	Compliant withAMBA® 5 AXI protocol	64-bit	Provides access to memory and peripheral components in the system.
Instruction Tightly Coupled Memory (ITCM) and *Data Tightly Coupled Memory* (DTCM) interfaces	-	• ITCM: 64-bit • DTCM: 4 banks of 32-bits	One ITCM interface and four DTCM interfaces to provide high-bandwidth access from the Cortex®-M85 processor and *Slave AHB* (S-AHB) interface to local low-latency memory. The size of both TCM instances is configurable, and in the range of 4KB-16MB in powers of 2. The Cortex®-M85 processor also supports zero size TCMs.
S-AHB interface	AMBA® 5 AHB	64-bit	Provides system access to the TCMs. A *Direct Memory Access* (DMA) engine typically uses this interface.
Tightly coupled master *Peripheral AHB* (P-AHB) interface	AMBA® 5 AHB	32-bit	Provides access to system peripherals.
Core *External Private Peripheral Bus* (EPPB) interface	AMBA® 4 APB	32-bit	Used to connect to external peripherals, including EWIC.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Name	Protocol	Width	Details
Debug *External Private Peripheral Bus* (EPPB) interface	AMBA® 4 APB	32-bit	Used to connect to external peripherals, for connecting to CoreSight™-compliant peripherals, TPIU, ETB, MCU Rom Table and other external debug peripherals.
External IDAU interface	-	-	Allows the system to define security attributes.
ITM and ETM interfaces	AMBA® 4 ATB	8-bit	Provides tracing capability.
Coprocessor interface	-	64-bit	Used for closely-coupled external accelerator hardware.
Debug AHB (D-AHB) interface	AMBA® 5 AHB slave	32-bit	Provides debug access to registers, memory, and peripherals.
Cross Trigger Interface (CTI) interface	-	Four channels	Used for debug and trace synchronization.
Power control interface	P-Channel and Q-Channel	-	Optional support for a number of internal power domains which can be enabled and disabled using the P-Channel and Q-Channel interfaces connected to a power controller in the system. For more information, see 7 Power management on page 115 or the *Arm® Cortex®-M85 Processor Integration and Implementation Manual* . The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is only available to licensees.
External Wakeup Interrupt Controller (EWIC) interface	-	-	Provides access to an optional EWIC, which is a peripheral to the system and is suitable for sleep states where the entire processor sub-system is powered down.

3.3 Security

Arm® TrustZone® technology uses the Security Extension, which supports Secure and Non-secure states on all memory interfaces, including security gating on *Tightly Coupled Memory* (TCM) interfaces.

Memory and peripherals in the system can be marked as Secure, making them accessible only to code that is running in the Secure state.

Interrupts can be marked as Secure indicating that they are handled by Secure handler code in the Secure world.

Hardware protects all Secure resources, including firmware and sensitive data values from being visible to Non-secure code and debug. If you are programming in Secure state, you can choose which Secure functions can be called by Non-secure code, where the Secure functions can tightly control the parameters of such function calls.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4 Reliability

The following are the Cortex®-M85 processor reliability features.

- L1 cache and TCM interfaces support optional internal *Error Correcting Code* (ECC). All ECC errors are reported to the system on an external interface.
- *Reliability, Availability, and Serviceability* (RAS) Extension support.

3.5 Power intent

The Cortex®-M85 processor power intent features include:

- Support for multiple power domain *State Retention Power Gating* (SRPG) implementation through *Unified Power Format* (UPF). The UPF files are IEEE 1801-2009 compliant.
- Power control based on the Arm standard P-Channel and Q-Channel interfaces. For information on the P-Channel and Q-Channel logic interfaces, see *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.
- Support for an *Internal Wakeup Interrupt Controller* (IWIC) and an *External Wakeup Interrupt Controller* (EWIC).

3.6 Cortex®-M85 implementation options

The Cortex®-M85 processor has configurable options that the chip designer can set during the implementation and integration stages to match your functional requirements.

The following table shows the Cortex®-M85 processor configurable option available at implementation time.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 3-3: Cortex®-M85 processor configurable options

Feature	Options
Floating-point and *M-profile Vector Extension* (MVE) support	The floating-point and MVE features together specify the MVE functionality that is supported on the Cortex®-M85 processor. Floating-point functionality can either be included or excluded. If floating-point functionality is not included, then the MVE options can be either of the following: • MVE not included • Integer subset of MVE included If floating-point functionality is included, then half-precision, single-precision, and double-precision floating-point operation is supported. The MVE options can be any of the following: • MVE not included • Integer, half-precision, and single-precision floating-point MVE are included **Note:** All other parameter combinations are invalid.
Coprocessor support	No support for coprocessor hardware
	Support for coprocessor hardware
Inclusion of Non-secure *Memory Protection Unit* (MPU)	0 region, 4 regions, 8 regions, 12 regions, or 16 regions
Inclusion of Secure *Memory Protection Unit* (MPU)	0 region, 4 regions, 8 regions, 12 regions, or 16 regions
Inclusion of *Security Attribution Unit* (SAU)	0 region, 4 regions, or 8 regions
Inclusion and size of instruction cache	No *Instruction Cache Unit* (ICU)
	ICU included, and the size can be 4KB, 8KB, 16KB, 32KB, or 64KB.
Inclusion and size of data cache	Area optimized M-AXI interface, no *Data Cache Unit* (DCU)
	DCU included and the size can be 4KB, 8KB, 16KB, 32KB, or 64KB
Inclusion of *Error Correcting Code* (ECC)	No ECC on caches or TCMs
	ECC on all implemented caches and TCMs
Number of interrupts	1-480 interrupts. To support non-contiguous mapping, you can remove individual interrupts.
Number of exception priority bits	3-8 priority bits.
Disable support for individual interrupts	When set to 1, support for individual interrupts is disabled, therefore, allowing a range of non-contiguous interrupts.
Debug resources included. This feature also controls the number of *Performance Monitoring Unit* (PMU) counters that are present.	Reduced set. Four data watchpoint comparators and four breakpoint comparators.
	Full set. Eight data watchpoint comparators and eight breakpoint comparators.
Inclusion of *Instrumentation Trace Macrocell* (ITM) and *Data Watchpoint and Trace* (DWT) trace	No ITM or DWT trace
	Complete ITM and DWT trace
Inclusion of *Embedded Trace Macrocell* (ETM)	No ETM support
	ETM instruction execution trace
Inclusion of *Internal Wakeup Interrupt Controller*	No IWIC
	IWIC is included

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Feature	Options
Number of IRQ lines supported by the IWIC and EWIC	The value always includes the three internal events NMI, RXEV, **EDBGRQ**, and at least one IRQ.
Inclusion of ITCM security gating	No ITCM security gate
	ITCM security gate included
ITCM security gate block size in bytes	$2^{(\text{Instruction } \textit{TCM Gate Unit} \text{ (TGU) block size+5})}$.
Number of ITCM security gate blocks	$2^{\text{Maximum number of instruction TGU blocks}}$
Inclusion of DTCM security gating	No DTCM security gate
	DTCM security gate included
DTCM security gate block size in bytes	$2^{(\text{Data TGU block size+5})}$
Number of DTCM security gate blocks	$2^{\text{Maximum number of data TGU blocks}}$
Pointer Authentication and Branch Target Identification Extension (PACBTI) extension support	No PACBTI extension present
	PACBTI extension present
Reset all registers functionality	Specifies whether all synchronous states or only the architecturally required states are reset. • Only reset states that architecture requires. • Reset all synchronous states.

Note

- The parameter to control inclusion of the *External Wakeup Interrupt Controller* (EWIC) can be configured at the MCU level. The MCU level supports all the processor-level configuration and contains additional configuration parameters to configure the functionality that is specific to CoreSight™ components that are included in the system.
- Signal tie-offs determine the inclusion of the ITCM and DTCM.
- Additionally, there are static and reset configuration signals. For more information, see C.3 Static configuration signals on page 342 and C.4 Reset configuration signals on page 343.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4 Programmers model

This chapter describes the Cortex®-M85 processor register set, modes of operation, and provides information on programming the Cortex®-M85 processor.

The Cortex®-M85 programmers model is an implementation of the Main Extension architecture. For a complete description of the programmers model, see the *Arm®v8-M Architecture Reference Manual*.

4.1 Security states, operation, and execution modes

The Cortex®-M85 processor supports Secure and Non-secure Security states, Thread and Handler operating modes, and can run in either Thumb or Debug operating states. In addition, the Cortex®-M85 processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

See the *Arm®v8-M Architecture Reference Manual* for more information about the modes of operation and execution.

Security states

The programmers model includes two orthogonal Security states, Secure state and Non-secure state. This means the processor is in Secure state or Non-secure state, but not both at the same time. The Cortex®-M85 processor always resets into Secure state. Each Security state includes a set of independent operating modes and supports both privileged and unprivileged user access. Registers in the *System Control Space* (SCS) are banked across Secure and Non-secure state, with the Non-secure register view available at an aliased address to Secure state.

Operating modes

For each Security state, the Cortex®-M85 processor can operate in Thread or Handler mode. The conditions which cause the Cortex®-M85 processor to enter Thread or Handler mode are as follows:

- The Cortex®-M85 processor enters Thread mode on reset, or as a result of an exception return to Thread mode. The Thread mode supports both privileged and unprivileged execution.
- The Cortex®-M85 processor enters Handler mode as a result of an exception. The Handler mode only supports privileged execution.

The Cortex®-M85 processor can change Security state on taking an exception. For example, when a Secure exception is taken from Non-secure state Thread or Handler mode, the Cortex®-M85 processor enters the Secure state Handler mode.

The Cortex®-M85 processor can also call Secure functions from Non-secure state and Non-secure functions from Secure state. The Security Extension includes requirements for these calls to prevent secure data from being accessed in Non-secure state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Operating states

The Cortex®-M85 processor can operate in T32 or Debug state:

- T32 state is the state of normal execution running 16-bit and 32-bit halfword-aligned T32 instructions.
- Debug state is the state when the Cortex®-M85 processor is in Halting debug.

Privileged access and unprivileged user access

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

4.2 Instruction set summary

The Cortex®-M85 processor implements the Arm®v8.1-M instruction set.

These instructions include:

- All base instructions
- All instructions in the Main Extension
- All instructions in the *Digital Signal Processing* (DSP) Extension
- Optionally, some of the coprocessor instructions. These are:
  - `CDP`, `CDP2`
  - `MCR`, `MCR2`
  - `MCRR`, `MCRR2`
  - `MRC`, `MRC2`
  - `MRRC`, `MRRC2`
- All instructions in the Security Extension
- Optionally, all half-precision, single-precision, and double-precision instructions in the Floating-point Extension
- Optionally, all vector operation instructions on integer operations in the *M-profile Vector Extension* (MVE)
- Optionally, all vector operation instructions on half-precision and single-precision floating-point operations in MVE
- Optionally, all the *Reliability, Availability, and Serviceability* (RAS) Extension instructions
- Optionally, all instructions added by the *Pointer Authentication and Branch Target Identification* (PACBTI) Extension

For more information about these instructions, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3 Exclusive monitor

The Cortex®-M85 processor implements a local exclusive monitor contained in the *Load Store Unit* (LSU). The local monitor within the Cortex®-M85 processor has been constructed not to hold any physical address, but instead treats any store-exclusive access as matching the address of the previous load-exclusive.

This means that the implemented exclusives reservation granule is the entire memory address range. The TCMs support the local exclusive monitor, but not shared or global exclusive monitors. This implies that the TCMs support exclusive requests between threads running on the Cortex®-M85 processor, but not exclusive requests between the Cortex®-M85 processor and a DMA (through the S-AHB).

If an exclusive read access is carried out to a region which does not support a global monitor it must respond accordingly with either **HEXOKAY** LOW or **RRESP[1:0]** OKAY.

These responses result in the transaction completing without setting the internal exclusive monitor. A subsequent exclusive store instruction does not carry out any memory transactions and sets the destination register to 1 indicating the exclusive access failed.

The external bus interfaces support an external exclusive monitor in the system to be shared with other bus masters.

For more information about semaphores and the local exclusive monitor, see the *Arm®v8-M Architecture Reference Manual*.

4.4 Cortex®-M85 processor core registers summary

The Cortex®-M85 processor core registers are 32 bits wide.

Some of the registers are banked. The Secure view of these registers is available when the processor is in Secure state. The Non-secure view is available when the processor is in Non-secure and Secure state.

The following table shows the processor core register set summary. See the *Arm®v8-M Architecture Reference Manual* for information about the Cortex®-M85 processor core registers and their addresses, access types, and reset values.

Table 4-1: Processor core register set summary

Name	Description
R0-R12	R0-R12 are general-purpose registers for data operations.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Name	Description
MSP (R13) PSP (R13)	The stack pointer, SP, is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, main stack pointer, MSP, or process stack pointer, PSP. There are two MSP registers in the Cortex®-M85 processor: • MSP_NS for the Non-secure state • MSP_S for the Secure state There are two PSP registers in the Cortex®-M85 processor: • PSP_NS for the Non-secure state • PSP_S for the Secure state
MSPLIM PSPLIM	The stack limit registers limit the extent to which the MSP and PSP registers respectively can descend. There are two MSPLIM registers in the Cortex®-M85 processor: • MSPLIM_NS for the Non-secure state • MSPLIM_S for the Secure state There are two PSPLIM registers in the Cortex®-M85 processor: • PSPLIM_NS for the Non-secure state • PSPLIM_S for the Secure state
LR (R14)	The Link Register, LR, is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	The Program Counter, PC, is register R15. It contains the current program address.
XPSR	The Program Status Register, XPSR, combines: • Application Program Status Register, APSR • Interrupt Program Status Register, IPSR • Execution Program Status Register, EPSR These registers provide different views of the XPSR.
PRIMASK	The PRIMASK register prevents activation of exceptions with configurable priority. For information about the Exception model the Cortex®-M85 processor supports, see 4.6 Exceptions on page 46. There are two PRIMASK registers in the Cortex®-M85 processor: • PRIMASK_NS for the Non-secure state • PRIMASK_S for the Secure state
BASEPRI	The BASEPRI register defines the minimum priority for exception processing. There are two BASEPRI registers in the Cortex®-M85 processor: • BASEPRI_NS for the Non-secure state • BASEPRI_S for the Secure state
FAULTMASK	The FAULTMASK register prevents activation of all exceptions except for non-maskable interrupt, NMI and optionally Secure HardFault. There are two FAULTMASK registers in the Cortex®-M85 processor: • FAULTMASK_NS for the Non-secure state • FAULTMASK_S for the Secure state
LO_BRANCH_INFO	Loop and branch tracking information. Software cannot access LO_BRANCH_INFO.
SP	Current stack pointer register. SP_NS for the Non-secure stack pointer register.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Name	Description
FPSCR	Floating-point Status and Control Register
S0-S31 / D0-15 / Q0-Q7	S0-S31 are 32 single-precision floating-point registers. These can also be treated as: • 16 double-precision floating-point registers (D0-D15) • 8 vector registers (Q0-Q7) The *Extension Processing Unit* (EPU) can be configured to perform floating-point and *M-profile Vector Extension* (MVE) operations. See 14 Floating-point and MVE support on page 224
VPR	Vector Predication Status and Control Register
CONTROL	The CONTROL register controls the stack that is used, and optionally the privilege level, when the Cortex®-M85 processor is in Thread mode. There are two CONTROL registers in the Cortex®-M85 processor: • CONTROL_NS for the Non-secure state • CONTROL_S for the Secure state.
PAC_KEY	Eight pointer authentication key registers

4.5 Architectural registers

Architectural registers can be either fully architectural or architectural with some **IMPLEMENTATION DEFINED** bit fields.

5 System registers on page 49 summarizes the Cortex®-M85 processor architectural registers as follows:

- 5.1 System control register summary on page 49
- 5.2 Identification register summary on page 52
- 5.5 Cache identification register summary on page 59

In each summary table, the description column contains either the name of each fully architectural register or a link to the definition of architectural registers that have **IMPLEMENTATION DEFINED** bit fields.

For details on fully architectural registers, see the *Arm®v8-M Architecture Reference Manual*.

4.6 Exceptions

Exceptions are handled and prioritized by the Cortex®-M85 processor and the *Nested Vectored Interrupt Controller* (NVIC). In addition to architecturally defined behavior, the Cortex®-M85 processor implements advanced exception and interrupt handling that reduces interrupt latency and includes **IMPLEMENTATION DEFINED** behavior.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.6.1 Exception handling and prioritization

The Cortex®-M85 processor core and the *Nested Vectored Interrupt Controller* (NVIC) together prioritize and handle all exceptions.

When handling 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 Cortex®-M85 processor supports tail-chaining that enables back-to-back interrupts without the overhead of state saving and restoration.

SoC designers configure the number of interrupts and bits of interrupt priority, 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.

Exceptions can be programmed as either Secure or Non-secure. When an exception is taken, the Cortex®-M85 processor switches to the associated Security state. The priority of Secure and Non-secure exceptions can be programmed independently. It is possible to deprioritize Non-secure configurable exceptions using AIRCR.PRIS to enable Secure interrupts to take priority. When taking and returning from an exception, the register state is always stored using the stack pointer associated with the background Security state. When taking a Non-secure exception from Secure state, all the register states are stacked, and then the registers are cleared to prevent Secure data being available to the Non-secure handler. The vector table base address is banked between Secure and Non-secure state. VTOR_S, contains the Secure vector table base address and VTOR_NS contains the Non-secure vector table base address. These registers can be programmed by software and also initialized at reset by the system.

Vector table entries are compatible with interworking between Arm and Thumb® instructions. This causes bit[0] of the vector value to load into EPSR.T, on exception entry. All populated vectors in the vector table entries must have bit[0] set. Creating a vector table entry with bit[0] clear generates an INVSTATE (Invalid state flag) fault on the first instruction of the handler corresponding to this vector.

Input signals **INITSVTOR[31:7]** and **INITNSVTOR[31:7]** initialize the Secure and Non-secure vector table base address, respectively.

Note

- The Cortex®-M85 processor abandons multicycle instructions to take pending interrupts. However, the processor supports resumption of a large subset of these multicycle instructions.
- Load Multiple and Store Multiple operations can be interrupted and resumed.
- Device accesses might cause an increase in interrupt latency. External Debugger accesses might also delay interrupt entry.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.6.2 Multicycle instructions

A multicycle instruction can take one or more clock cycles to complete.

Load Multiple and Store Multiple operations are examples of a multicycle instruction.

Note

A single load that is issued but waiting (wait stated) on the bus is not considered a multicycle instruction. Therefore it is not abandoned to take pending interrupts.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5 System registers

This chapter describes the system registers for the Cortex®-M85 processor.

5.1 System control register summary

The system control registers are a combination of fully architectural and **IMPLEMENTATION DEFINED** 32-bit registers and can be set to control various processor features.

The following table shows a summary of the system control registers.

For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

Table 5-1: System control register summary

Address	Name	Type	Reset value	Description
`0xE000ECFC`	REVIDR	RO	`0x00000000` **Note:** The value of REVIDR[3:0] is determined by the input signal **REVIDRNUM** as specified in C.28 Miscellaneous signals on page 368	5.6 REVIDR, Revision ID Register on page 63
`0xE000ED00`	CPUID	RO	`0x410FD232`	5.4 CPUID, CPUID Base Register on page 58
`0xE000ED04`	ICSR	RW	`0x00000000`	Interrupt Control and State Register
`0xE000ED08`	VTOR	RW	`0xXXXXXXX0` **Note:** Bits [31:7] of VTOR_S are based on **INITSVTOR[31:7]**. Bits [31:7] of VTOR_NS are based on **INITNSVTOR[31:7]**. Bits [6:0] are **RES0**.	Vector Table Offset Register
`0xE000ED0C`	AIRCR	RW	`0xFA05X000` **Note:** Bit [15] of this register depends on input signal **CFGBIGEND**. Bits [14:0] reset to zero.	Application Interrupt and Reset Control Register
`0xE000ED10`	SCR	RW	`0x00000000`	System Control Register
`0xE000ED14`	CCR	RW	`0x00000201`	Configuration and Control Register
`0xE000ED18`	SHPR1	RW	`0x00000000`	System Handler Priority Register 1

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE000ED1C	SHPR2	RW	0x00000000	System Handler Priority Register 2
0xE000ED20	SHPR3	RW	0x00000000	System Handler Priority Register 3
0xE000ED24	SHCSR	RW	0x00000000	System Handler Control and State Register
0xE000ED28	CFSR	RW	0x00000000	Configurable Fault Status Register A 32-bit register comprising MMFSR, BFSR, and UFSR
	MMFSR	RW	0x00	MemManage Fault Status Register
0xE000ED29	BFSR	RW	0x00	BusFault Status Register
0xE000ED2A	UFSR	RW	0x0000	UsageFault Status Register
0xE000ED2C	HFSR	RW	0x00000000	HardFault Status Register
0xE000ED30	DFSR	RW	0x00000000 Cold reset only.	Debug Fault Status Register
0xE000ED34	MMFAR	RW	0x00000000 Only when RAR is enabled, otherwise **UNKNOWN**.	MemManage Fault Address Register
0xE000ED38	BFAR	RW	0x00000000 Only when RAR is enabled, otherwise **UNKNOWN**.	BusFault Address Register
0xE000ED3C	AFSR	RW	0x00000000	5.3 AFSR, Auxiliary Fault Status Register on page 56

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE000ED40	ID_PFR0	RO	0x20000030 **Note:** ID_PFR0[31:28] indicates support for the RAS Extension. ID_PFR0[31:28] is 0b0010 indicating that version 1 is implemented.	5.18 ID_PFR0, Processor Feature Register 0 on page 96
0xE000ED44	ID_PFR1	RO	0x00000230 **Note:** ID_PFR1[7:4] is 0b0011, which indicates support for the Security Extension.	Processor Feature Register 1
0xE000ED48	ID_DFR0	RO	0x10X00000 **Note:** ID_DFR0[23:20] indicates support for debug architecture. If halting debug is implemented and either a reduced set or a full set of debug resources is configured, then ID_DFR0[23:20] is 0b0010.	Debug Feature Register 0
0xE000ED4C	ID_AFR0	RO	0x00000000	Auxiliary Feature Register 0
0xE000ED50	ID_MMFR0	RO	0x00111040 **Note:** ID_MFR0[23:20] indicates support of Auxiliary Control registers. ID_MFR0[19:16] indicates support of TCMs. ID_MFR0[15:12] indicates that two levels of Shareability are implemented. ID_MFR0[11:8] indicates that the Outermost Shareability is implemented as Non-cacheable. ID_MFR0[7:4] indicates PMSAv8 support. All other bits are **RES0**.	Memory Model Feature Register 0
0xE000ED54	ID_MMFR1	RO	0x00000000	Memory Model Feature Register 1
0xE000ED58	ID_MMFR2	RO	0x01000000 **Note:** ID_MFR2[27:24] indicates that WFI can stall. All other bits are **RES0**.	Memory Model Feature Register 2
0xE000ED5C	ID_MMFR3	RO	0x00000011 **Note:** ID_MFR3[11:8] indicates that branch predictor invalidate is not supported. ID_MFR3[7:4] indicates that set/way maintenance operations are supported. ID_MFR3[3:0] indicates that address and instruction cache invalidate maintenance operations are supported. All other bits are **RES0**.	Memory Model Feature Register 3
0xE000ED60	ID_ISAR0	RO	0x011X3110 ID_ISAR0[19:16] depend on whether the external coprocessor interface is included in the processor. • If the external coprocessor is not included, there is no coprocessor instruction support, except the FPU. The value of X is 0x0. • If the external coprocessor is included, coprocessor instruction support is included. The value of X is 0x4.	Instruction Set Attribute Register 0
0xE000ED64	ID_ISAR1	RO	0x02212000	Instruction Set Attribute Register 1
0xE000ED68	ID_ISAR2	RO	0x20232232	Instruction Set Attribute Register 2
0xE000ED6C	ID_ISAR3	RO	0x01111131	Instruction Set Attribute Register 3

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE000ED70`	ID_ISAR4	RO	`0x01310132`	Instruction Set Attribute Register 4
`0xE000ED74`	ID_ISAR5	RO	`0x00000000`	Instruction Set Attribute Register 5
`0xE000ED78`	CLIDR	RO	`0xXXX0000X` **Note:** CLIDR[31:21] and CLIDR[2:0] depend on the cache configuration of the processor.	5.5.1 CLIDR, Cache Level ID Register on page 60
`0xE000ED7C`	CTR	RO	• If an instruction cache or data cache is included, then the reset value is `0x8303C003`. • If an instruction cache or data cache is not included, then the reset value is `0x00000000`.	Cache Type Register
`0xE000ED80`	CCSIDR	RO	`0xXXXXXXXX` **Note:** CCSIDR depends on the CSSELR setting and L1 cache configuration.	5.5.3 CCSIDR, Current Cache Size ID Register on page 62
`0xE000ED84`	CSSELR	RW	`0x00000000`	5.5.2 CSSELR, Cache Size Selection Register on page 61
`0xE000ED88`	CPACR	RW	`0x00000000`	Coprocessor Access Control Register
`0xE000ED8C`	NSACR	RW	`0x00000000`	Non-secure Access Control Register

5.2 Identification register summary

The Cortex®-M85 processor identification registers allow software to determine the features and functionality that are available. Each of these registers is 32 bits wide.

The following table shows a summary of the identification registers. For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

Table 5-2: Identification register summary

Address	Name	Type	Reset value	Description
`0xE000ED00`	CPUID	RO	`0x410FD232`	5.4 CPUID, CPUID Base Register on page 58

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE000ED40`	ID_PFR0	RO	`0x20000030` **Note:** ID_PFR0[31:28] indicates support for the RAS Extension. ID_PFR0[31:28] is `0b0010` indicating that version 1 is implemented.	Processor Feature Register 0
`0xE000ED44`	ID_PFR1	RO	`0x00000230` **Note:** ID_PFR1[7:4] indicates support for the Security Extension.	Processor Feature Register 1
`0xE000ED48`	ID_DFR0	RO	`0x10200000` **Note:** ID_DFR0[23:20] indicates support for debug architecture. If halting debug is implemented and either a reduced set or a full set of debug resources is configured, then ID_DFR0[23:20] is `0b0010`.	Debug Feature Register 0
`0xE000ED4C`	ID_AFR0	RO	`0x00000000`	Auxiliary Feature Register 0
`0xE000ED50`	ID_MMFR0	RO	`0x00111040` **Note:** ID_MFR0[23:20] indicates support of Auxiliary Control registers. ID_MFR0[19:16] indicates support of TCMs. ID_MFR0[15:12] indicates that two levels of Shareability are implemented. ID_MFR0[11:8] indicates that the Outermost Shareability is implemented as Non-cacheable. ID_MFR0[7:4] indicates PMSAv8 support. All other bits are **RES0**.	Memory Model Feature Register 0
`0xE000ED54`	ID_MMFR1	RO	`0x00000000`	Memory Model Feature Register 1
`0xE000ED58`	ID_MMFR2	RO	`0x01000000` **Note:** ID_MFR2[27:24] indicates that WFI can stall. All other bits are **RES0**.	Memory Model Feature Register 2
`0xE000ED5C`	ID_MMFR3	RO	`0x00000011` **Note:** ID_MFR3[11:8] indicates that branch predictor invalidate is not supported. ID_MFR3[7:4] indicates that set/way maintenance operations are supported. ID_MFR3[3:0] indicates that address and instruction cache invalidate maintenance operations are supported. All other bits are **RES0**.	Memory Model Feature Register 3
`0xE000ED60`	ID_ISAR0	RO	`0x011X3110` ID_ISAR0[19:16] depend on whether the external coprocessor interface is included in the processor. • If the external coprocessor is not included, there is no coprocessor instruction support, except the FPU. The value of `X` is `0x0`. • If the external coprocessor is included, coprocessor instruction support is included. The value of `X` is `0x4`.	Instruction Set Attributes Register 0
`0xE000ED64`	ID_ISAR1	RO	`0x02212000`	Instruction Set Attributes Register 1
`0xE000ED68`	ID_ISAR2	RO	`0x20232232`	Instruction Set Attributes Register 2
`0xE000ED6C`	ID_ISAR3	RO	`0x01111131`	Instruction Set Attributes Register 3

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE000ED70	ID_ISAR4	RO	0x01310132	Instruction Set Attributes Register 4
0xE000ED74	ID_ISAR5	RO	0x00100000 **Note:** ID_ISAR5[23:20] indicates the existence and type of address authentication algorithm. It reads as 0b0001, indicating that address authentication using the QARMA algorithm is implemented.	Instruction Set Attributes Register 5
0xE000ED78	CLIDR	RO	0xXXX0000X **Note:** Bits CLIDR[31:21] and CLIDR[2:0] depend on the cache configuration of the processor.	5.5.1 CLIDR, Cache Level ID Register on page 60
0xE000ED7C	CTR	RO	• If an instruction cache or data cache is included, then the reset value is 0x8303C003. • If an instruction cache or data cache is not included, then the reset value is 0x00000000.	Cache Type Register
0xE000ED80	CCSIDR	RO	0xXXXXXXXX **Note:** CCSIDR depends on the CSSELR setting and L1 cache configuration.	5.5.3 CCSIDR, Current Cache Size ID Register on page 62
0xE000ED84	CSSELR	RW	0x00000000	5.5.2 CSSELR, Cache Size Selection Register on page 61
0xE000EF40	MVFR0	RO	Table 5-3: MVFR0, MVFR1, and MVFR2 reset values on page 56	Media and VFP Feature Register 0
0xE000EF44	MVFR1	RO		Media and VFP Feature Register 1
0xE000EF48	MVFR2	RO		Media and VFP Feature Register 2
0xE000EFD0	DPIDR4	RO	0x00000004	CoreSight™ Peripheral ID Register 4
0xE000EFD4	DPIDR5	RO	0x00000000	CoreSight™ Peripheral ID Register 5
0xE000EFD8	DPIDR6	RO	0x00000000	CoreSight™ Peripheral ID Register 6
0xE000EFDC	DPIDR7	RO	0x00000000	CoreSight™ Peripheral ID Register 7
0xE000EFE0	DPIDR0	RO	0x00000023	CoreSight™ Peripheral ID Register 0

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE000EFE4	DPIDR1	RO	0x000000BD	CoreSight™ Peripheral ID Register 1
0xE000EFE8	DPIDR2	RO	0x0000000B	CoreSight™ Peripheral ID Register 2
0xE000EFEC	DPIDR3	RO	0x00000000 **Note:** Bits [7:4] and [3:0] are REVAND and CMOD respectively. The REVAND field indicates minor errata fixes specific to this design, for example metal fixes after implementation. If the component is reusable IP, the CMOD field indicates whether you have modified the behavior of the component. These values depend on the exact revision of the silicon as documented in *Arm® CoreSight™ Architecture Specification v3.0*.	CoreSight™ Peripheral ID Register 3
0xE0005E10	ERRIIDR	RO	0x0xD230043B	Error Implementer ID Register
0xE000EFF0	DCIDR0	RO	0x0000000D	CoreSight™ Component ID Register 0
0xE000EFF4	DCIDR1	RO	0x00000090	CoreSight™ Component ID Register 1
0xE000EFF8	DCIDR2	RO	0x00000005	CoreSight™ Component ID Register 2
0xE000EFFC	DCIDR3	RO	0x000000B1	CoreSight™ Component ID Register 3
0xE000EFBC	DDEVARCH	RO	0x47702A04	CoreSight™ Device Architecture Register
0xE000ECFC	REVIDR	RO	0x00000000 **Note:** The value of REVIDR[3:0] is determined by the input signal **REVIDRNUM** as specified in C.28 Miscellaneous signals on page 368	5.6 REVIDR, Revision ID Register on page 63
0xE0005FC8	ERRDEVID	RO	0x00000001 **Note:** ERRDEVID[15:0] indicates the number of error records that the RAS Extension implementation supports. In the Cortex®-M85 processor, this field reads 0x0001 indicating one error record is supported. This register is RAZ if any of the following conditions are true: • ECC protection is not configured. • ECC protection is configured but not enabled	11.4.6 ERRDEVID, RAS Error Record Device ID Register on page 215.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.2.1 Media and VFP Feature Register reset values, MVFR0, MVFR1, and MVFR2 reset values

The MVFR0, MVFR1, and MVFR2 register reset values depend on the *M-profile Vector Extension* (MVE) and floating-point functionality configuration. The MVE and floating-point functionality operation is configured using the `MVE` and `FPU` configuration parameters.

For more information, see 3.6 Cortex-M85 implementation options on page 39.

The following table shows the MVFR0, MVFR1, and MVFR2 reset values based on the reset configurations.

Table 5-3: MVFR0, MVFR1, and MVFR2 reset values

Configuration	MVFR0	MVFR1	MVFR2
`MVE`=0, `FPU`=0	0x00000000	0x00000000	0x00000000
`MVE`=1, `FPU`=0	0x00000001	0x00000100	0x00000000
`MVE`=0, `FPU`=1	0x10110221	0x12100011	0x00000040
`MVE`=2, `FPU`=1	0x10110221	0x12100211	0x00000040

5.3 AFSR, Auxiliary Fault Status Register

The AFSR provides fault status information.

Usage constraints

Privileged access permitted only. Unprivileged accesses generate a fault. The register is set to zero at reset. A field in the register can be cleared by writing `0b1` to the corresponding bit. AFSR bits [31:21] are only valid if BFSR.IBUSERR is set. AFSR bits [20:10] are only valid if BFSR.PRECISEERR is set. AFSR bits [9:0] are only valid if BFSR.IMPRECISEERR is set. If multiple faults occur, the AFSR indicates the types of all the faults that have occurred. For more information on BFSR, see the *Arm®v8-M Architecture Reference Manual*.

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception

Configuration

This register is always implemented.

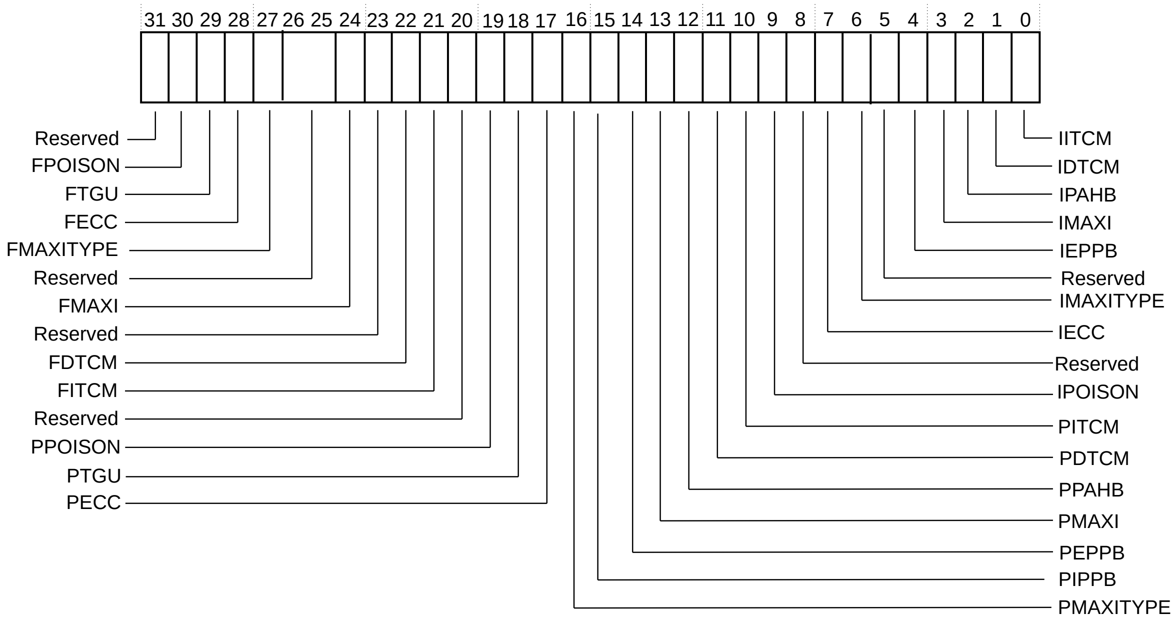
Attributes

A 32-bit RW register that is located at `0xE000ED3C`. Non-secure alias is provided using AFSR_NS, that is located at `0xE002ED3C`. This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the AFSR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-1: AFSR bit assignments



The following table describes the AFSR bit assignments.

Table 5-4: AFSR bit assignments

Bits	Name	Type	Description
[31]	Reserved	-	**RES0**
[30]	FPOISON	RW	Fetch fault that is caused by **RPOISON** or TEBRx.POISON.
[29]	FTGU	-	Fetch fault that is caused by *TCM Gate Unit* (TGU) security violation.
[28]	FECC	RW	Fetch fault that is caused by uncorrectable *Error Correcting Code* (ECC) error.
[27]	FMAXITYPE	RW	AXI response that caused the fetch fault. Only valid when AFSR.FMAXI is 1. `0b0` SLVERR `0b1` DECERR
[26:25]	Reserved	-	**RES0**
[24]	FMAXI	RW	Fetch fault on *Master AXI* (M-AXI) interface.
[23]	Reserved	-	**RES0**
[22]	FDTCM	RW	Fetch fault on *Data Tightly Coupled Memory* (DTCM) interface.
[21]	FITCM	RW	Fetch fault on *Instruction Tightly Coupled Memory* (ITCM) interface.
[20]	Reserved	-	**RES0**
[19]	PPOISON	RW	Precise fault that is caused by **RPOISON** or TEBRx.POISON.
[18]	PTGU	RW	Precise fault that is caused by TGU security violation.
[17]	PECC	RW	Precise fault that is caused by uncorrectable ECC error.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Description
[16]	PMAXITYPE	RW	AXI response that caused the precise fault. Only valid when AFSR.PMAXI is 1. `0b0` SLVERR `0b1` DECERR
[15]	PIPPB	RW	Precise fault on *Internal Private Peripheral Bus* (IPPB) interface.
[14]	PEPPB	RW	Precise fault on *External Private Peripheral Bus* (EPPB) interface.
[13]	PMAXI	RW	Precise fault on M-AXI interface.
[12]	PPAHB	RW	Precise fault on *Peripheral AHB* (P-AHB) interface.
[11]	PDTCM	RW	Precise fault on DTCM interface.
[10]	PITCM	RW	Precise fault on ITCM interface.
[9]	IPOISON	RW	Imprecise BusFault because of **RPOISON**.
[8]	Reserved	-	**RES0**
[7]	IECC	RW	Imprecise fault that is caused by uncorrectable ECC error.
[6]	IMAXITYPE	RW	AXI response that caused the imprecise fault. Only valid when AFSR.IMAXI is 1. `0b0` SLVERR `0b1` DECERR
[5]	Reserved	-	**RES0**
[4]	IEPPB	RW	Imprecise fault on EPPB interface.
[3]	IMAXI	RW	Imprecise fault on M-AXI interface.
[2]	IPAHB	RW	Imprecise fault on P-AHB interface.
[1]	IDTCM	RW	Imprecise fault on DTCM interface.
[0]	IITCM	RW	Imprecise fault on ITCM interface.

5.4 CPUID, CPUID Base Register

CPUID contains the Cortex®-M85 processor part number, version, and implementation information.

Usage constraints

This register is read-only.

Configuration

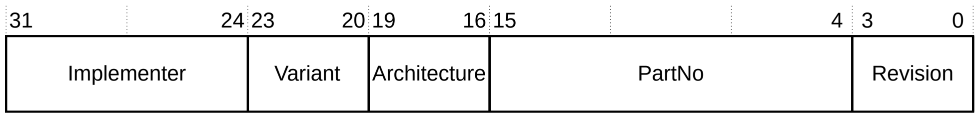
This register is always implemented.

Attributes

This register is not banked between Security states. See 5.2 Identification register summary on page 52 for more information.

The following figure shows the CPUID bit assignments.

Figure 5-2: CPUID bit assignments



The following table shows the CPUID bit assignments.

Table 5-5: CPUID bit assignments

Bits	Name	Type	Description
[31:24]	Implementer	RO	Implementer code that Arm has assigned. **`0x41`** A: Arm Limited.
[23:20]	Variant	RO	Variant number to distinguish between different product variants or major revisions of the product. Variant is the `x` in the `rxpy` product revision identifier. **`0x0`** Cortex®-M85 r0p2
[19:16]	Architecture	RO	Indicates the architecture version that the Cortex®-M85 processor implements. **`0b1111`** Arm®v8.1-M with Main Extension.
[15:4]	PartNo	RO	Part number of the Cortex®-M85 processor. **`0xD23`** Cortex®-M85
[3:0]	Revision	RO	Revision number to distinguish between different patches of the product. Revision is the `y` in the `rxpy` product revision identifier. **`0x2`** Cortex®-M85 r0p2

5.5 Cache identification register summary

The cache identification registers are responsible for cache configuration in the processor. The fields in these registers depend on the instruction and data cache size.

The following table lists the cache identification registers.

Table 5-6: Cache identification register summary

Address	Name	Type	Reset value	Description
`0xE000ED78`	CLIDR	RO	`0xXXX0000X` **Note:** CLIDR[31:21] and CLIDR[2:0] depend on the cache configuration of the processor.	5.5.1 CLIDR, Cache Level ID Register on page 60

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE000ED7C`	CTR	RO	• If an instruction cache or data cache is included, then the reset value is `0x8303C003`. • If an instruction cache or data cache is not included, then the reset value is `0x00000000`.	Cache Type Register. For more information, see the *Arm®v8-M Architecture Reference Manual*
`0xE000ED80`	CCSIDR	RO	`0xXXXXXXXX` **Note:** CCSIDR depends on the CSSELR setting and L1 cache configuration.	5.5.3 CCSIDR, Current Cache Size ID Register on page 62
`0xE000ED84`	CSSELR	RW	`0x00000000`	5.5.2 CSSELR, Cache Size Selection Register on page 61

5.5.1 CLIDR, Cache Level ID Register

The CLIDR identifies the type of caches that are implemented and the level of coherency and unification. If an instruction cache, data cache, or both is not configured in the processor, then CLIDR is `0x00000000`.

Usage constraints

This register is a read-only and is accessible in Privileged mode only.

Configuration

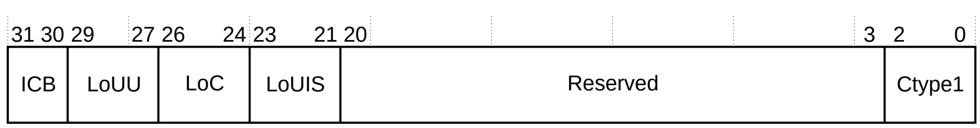
This register is always implemented.

Attributes

This register is not banked between Security states. See Table 5-2: Identification register summary on page 52 for more information.

The following figure shows the CLIDR bit assignments.

Figure 5-3: CLIDR bit assignments



The following table shows the CLIDR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 5-7: CLIDR bit assignments

Bits	Name	Type	Description
[31:30]	ICB	RO	Inner cache boundary. The Cortex®-M85 processor supports inner Cacheability on the bus. Therefore, this field cannot disclose any information. **0b00** Not disclosed in this mechanism.
[29:27]	LoUU	RO	Level of Unification Uniprocessor. The L1 cache must be cleaned or invalidated when cleaning or invalidating occurs to the point of unification. The options are: **0b000** Caches are not implemented. Therefore, cleaning and invalidation is not required. **0b001** *Level 1* (L1) data cache or instruction cache is implemented. Therefore, cleaning and invalidation are required.
[26:24]	LoC	RO	Level of Coherency. The L1 cache must be cleaned when cleaning occurs to the point of coherency. The options are: **0b000** Caches are not implemented. Therefore, cleaning is not required. **0b001** L1 data cache or instruction cache is implemented. Therefore, cleaning is required.
[23:21]	LoUIS	RO	Level of Unification Inner Shareable. The L1 cache must be cleaned or invalidated when cleaning or invalidating occurs to the point of unification for the inner Shareability domain. The options are: **0b000** Caches are not implemented. Therefore, cleaning and invalidation are not required. **0b001** L1 data cache or instruction cache is implemented. Therefore, cleaning and invalidation are required.
[20:3]	Reserved	-	**RES0**
[2:0]	Ctype1	RO	*Level 1* (L1) cache type. The options are: **0b000** Caches are not implemented. **0b001** Only instruction cache is implemented. **0b010** Only data cache is implemented. **0b011** Both data cache and instruction cache are implemented.

5.5.2 CSSELR, Cache Size Selection Register

The CSSELR selects the cache accessed through the CCSIDR by specifying the cache level and the type of cache (either instruction or data cache). For Cortex®-M85, this can be either the L1 instruction cache or L1 data cache.

Usage constraints

This register is read/write and is accessible in Privileged mode only.

Configurations

This register is always implemented.

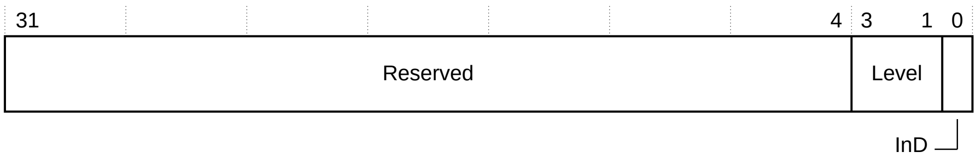
Attributes

See Table 5-2: Identification register summary on page 52 for more information.

This register is banked between Security states. The following figure shows the CSSELR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-4: CSSELR bit assignments



The following table shows the CSSELR bit assignments.

Table 5-8: CSSELR bit assignments

Bits	Name	Type	Function
[31:4]	Reserved	-	**RES0**
[3:1]	Level	RO	Identifies which cache level to select. **0x0** L1 cache. This field is RAZ/WI.
[0]	InD	RW	Selects either L1 instruction or data cache. The options are: **0** L1 data cache. **1** L1 instruction cache.

5.5.3 CCSIDR, Current Cache Size ID Register

The CCSIDR provides information about the architecture of the instruction or data cache that the CSSELR selects. If the cache corresponding to CSSELR.InD is not included in the processor, then this register reads `0x00000000`.

Usage constraints

This register is read-only and is accessible in Privileged mode only.

Configurations

This register is always implemented.

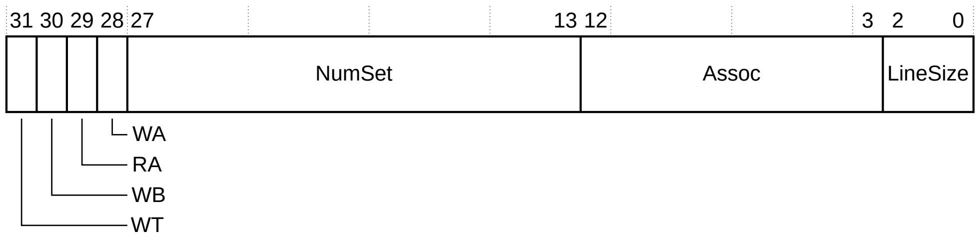
Attributes

This register is banked between Security states. The value of this register depends on the cache that CSSELR selects. If you are setting CSSELR in a particular Security state, then Arm recommends that you read CSSIDR in the same Security state to get the architecture information about the selected instruction or data cache.

The following figure shows the CCSIDR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-5: CCSIDR bit assignments



The following table shows the CCSIDR bit assignments.

Table 5-9: CCSIDR bit assignments

Bits	Name	Type	Function
[31]	WT	RO	Indicates support available for Write-Through: **0b1** Write-Through support available.
[30]	WB	RO	Indicates support available for Write-Back: **0b1** Write-Back support available.
[29]	RA	RO	Indicates support available for read allocation: **0b1** Read allocation support available.
[28]	WA	RO	Indicates support available for write allocation: **0b1** Write allocation support available.
[27:13]	NumSet	RO	Indicates the number of sets. Cache-size dependent.
[12:3]	Assoc	RO	Indicates associativity. The value depends on the cache that CSSELR selects. When CSSELR.InD=1 (L1 instruction cache): **0x1** 2-way set associative instruction cache. When CSSELR.InD=0 (L1 data cache): **0x3** 4-way set associative data cache.
[2:0]	LineSize	RO	Indicates the number of words in each cache line. **0b1** Represents 32 bytes.

The LineSize field is encoded as 2 less than log(2) of the number of words in the cache line. For example, a value of `0x0` indicates that there are four words in a cache line, that is the minimum size for the cache. A value of `0x1` indicates that there are eight words in a cache line.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.6 REVIDR, Revision ID Register

The REVIDR register provides additional **IMPLEMENTATION-SPECIFIC** minor revision that can be interpreted with the CPUID register.

Usage constraints

Unprivileged access results in a BusFault exception. This register is RAZ/WI from Non-secure state.

This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL_S.UIDAPEN or DAUTHCTRL_NS.UIDAPEN is set.

Configurations

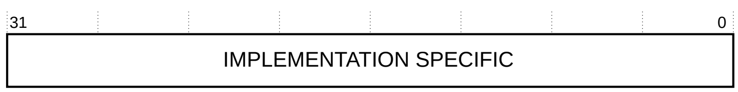
This register is always implemented.

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the REVIDR bit assignments.

Figure 5-6: REVIDR bit assignments



The following table describes the REVIDR bit assignments.

Table 5-10: REVIDR bit assignments

Field	Name	Type	Description
[31:0]	IMPLEMENTATION SPECIFIC	RO	**IMPLEMENTATION-SPECIFIC** minor revision information that can be interpreted with the CPUID register. For more information on the CPUID register, see the *Arm®v8-M Architecture Reference Manual*.

Note

The value of REVIDR[3:0] is determined by the input signal **REVIDRNUM** as specified in C.28 Miscellaneous signals on page 368

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.7 Implementation control register summary

Implementation control registers are architecturally defined with values that control aspects of system implementation.

The following table shows a summary of the implementation control registers. For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

Table 5-11: Implementation control register summary

Address	Name	Type	Reset value	Description
`0xE000E004`	ICTR	RO	`0x0000000X` **Note:** ICTR[3:0] depends on the number of interrupts that are included in the processor. Bits [31:4] are zero.	5.9 ICTR, Interrupt Controller Type Register on page 67
`0xE000E008`	ACTLR	RW	`0x00000000`	5.8 ACTLR, Auxiliary Control Register on page 65
`0xE000E00C`	CPPWR	RW	`0x00000000`	Coprocessor Power Control Register, see definition in the *Arm®v8-M Architecture Reference Manual*

5.8 ACTLR, Auxiliary Control Register

The ACTLR contains many fields that allow software to control processor features and functionality.

Usage constraints

Privileged access permitted only. Unprivileged accesses generate a BusFault exception.

Configuration

This register is always implemented.

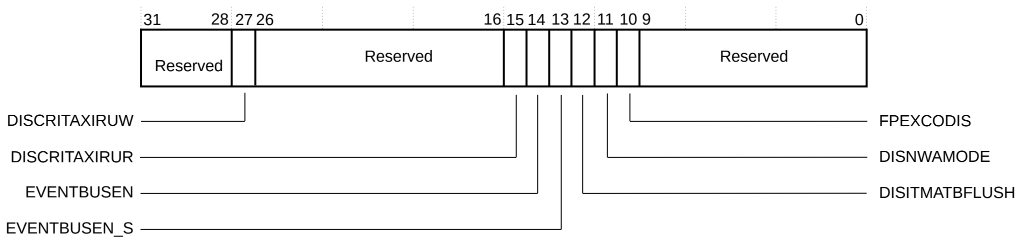
Attributes

A 32-bit RW register that is located at `0xE000E008`. This register is banked between Secure and Non-secure states to prevent non-secure software from altering the behavior of the secure state. At reset, all fields in this register are set to zero. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ACTLR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-7: ACTLR bit assignments



The following table describes the ACTLR bit assignments.

Table 5-12: ACTLR bit assignments

Bits	Name	Type	Description
[31:28]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.
[27]	DISCRITAXIRUW	RW	Disable-Critical-AXI-Read-Under-Write. The options for this bit are: **0** Normal operation **1** AXI reads to Device memory and exclusive reads to shared memory are not initiated on the M-AXI read address channel until all outstanding writes on the M-AXI interface are complete Setting this bit can decrease performance.
[26:16]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.
[15]	DISCRITAXIRUR	RW	Disable-Critical-AXI-Read-Under-Read. The options for this bit are: **0** Normal operation **1** AXI reads to Device memory and exclusive reads to shared memory are not initiated on the M-AXI read address channels until all outstanding writes on the M-AXI interface are complete Setting this bit can decrease performance.
[14]	EVENTBUSEN	RW	This bit activates EVENTBUS output. The options for this bit are: **0** EVENTBUS not active **1** EVENTBUS active This bit resets to 0 on a Warm reset. This bit is not banked.
[13]	EVENTBUSEN_S	RW	This bit activates Secure-only EVENTBUSEN. The options for this bit are: **0** EVENTBUSEN is accessible by both Security states **1** EVENTBUSEN is accessible by Secure state only This bit is RAZ/WI from a Non-Secure state. This bit resets to 0 on a Warm reset.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Description
[12]	DISITMATBFLUSH	RW	This bit determines whether *Instrumentation Trace Macrocell* (ITM) or *Data Watchpoint and Trace* (DWT) ATB flush is disabled. The options for this bit are: **0** Normal operation **1** ITM or DWT ATB flush disabled When disabled, the **AFVALID** signal (trace flush request) is ignored and the **AFREADY** (trace flush ready) signal is held HIGH.
[11]	DISNWAMODE	RW	This bit determines whether no write allocate mode is disabled. The options for this bit are: **0** Normal operation **1** No write allocate mode disabled Setting this bit decreases performance. For more information on no write allocation mode, see 10.9.1.1 No Write-Allocate mode on page 185.
[10]	FPEXCODIS	RW	This bit determines whether floating-point exception outputs are disabled. The options for this bit are: **0** Normal operation **1** Floating-point exception outputs disabled
[9:0]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.

5.9 ICTR, Interrupt Controller Type Register

The ICTR register shows the number of interrupt lines that the NVIC supports.

Usage Constraints

There are no usage constraints.

Configurations

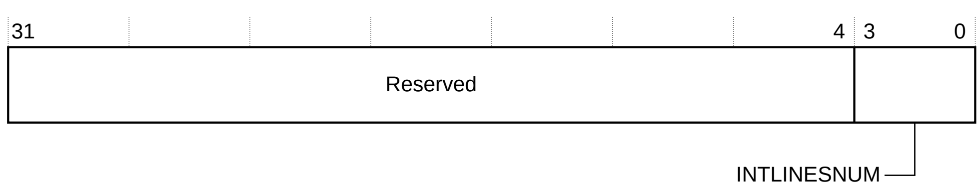
This register is available in all processor configurations.

Attributes

See 12.3 NVIC register summary on page 219 for more information.

The following figure shows the ICTR bit assignments.

Figure 5-8: ICTR bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table shows the ICTR bit assignments.

Table 5-13: ICTR bit assignments

Bits	Name	Type	Function
[31:4]	-	-	Reserved.
[3:0]	INTLINESNUM	RO	Total number of interrupt lines in groups of 32: **0b0000** 0-32 **0b0001** 33-64 **0b0010** 65-96 **0b0011** 97-128 **0b0100** 129-160 **0b0101** 161-192 **0b0110** 193-224 **0b0111** 225-256 **0b1000** 257-288 **0b1001** 289-320 **0b1010** 321-352 **0b1011** 353-384 **0b1100** 385-416 **0b1101** 417-448 **0b1110** 449-480

Note

The processor supports from 1 to 480 external interrupts.

5.10 IMPLEMENTATION DEFINED registers summary

The 32-bit **IMPLEMENTATION DEFINED** registers provide memory configuration and access control, error record information, interrupt control, and processor configuration information.

The following table lists the **IMPLEMENTATION DEFINED** registers for the Cortex®-M85 processor.

Table 5-14: IMPLEMENTATION DEFINED registers summary

Address	Name	Type	Reset value	Description
0xE0005000	ERRFR0	RO	0x00000101	11.4.1 ERRFR0, RAS Error Record Feature Register on page 208
0xE0005008	ERRCTRL0	-	-	This register is **RES0**.
0xE0005010	ERRSTATUS0	RW	0xXXX000XX	11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register on page 209
0xE0005018	ERRADDR0	RO	0xXXXXXXXX	11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers on page 212

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE000501C	ERRADDR20	RO	0xX0000000	11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers on page 212
0xE0005020	ERRMISC00	-	-	This register is **RES0**.
0xE0005024	ERRMISC10	RO	0x0000000X	11.4.4 ERRMISC10, Error Record Miscellaneous Register 10 on page 213
0xE0005028	ERRMISC20	-	-	This register is **RES0**.
0xE000502C	ERRMISC30	-	-	This register is **RES0**.
0xE0005030	ERRMISC40	-	-	This register is **RES0**.
0xE0005034	ERRMISC50	-	-	This register is **RES0**.
0xE0005038	ERRMISC60	-	-	This register is **RES0**.
0xE000503C	ERRMISC70	-	-	This register is **RES0**.
0xE0005E00	ERRGSR0	RO	0x00000000	11.4.5 ERRGSR0, RAS Fault Group Status Register on page 214
0xE000ECFC	REVIDR	RO	0x00000000	5.6 REVIDR, Revision ID Register on page 63
0xE0005FC8	ERRDEVID	RO	0x00000001	11.4.6 ERRDEVID, RAS Error Record Device ID Register on page 215
0xE000E008	ACTLR	RW	0x00000000	5.8 ACTLR, Auxiliary Control Register on page 65
0xE000ED3C	AFSR	RW	0x00000000	5.3 AFSR, Auxiliary Fault Status Register on page 56
0xE000EF04	RFSR	RW	0xXXXX000X	11.4.7 RFSR, RAS Fault Status Register on page 216
0xE001E000	MSCR	RW	If the instruction cache and data cache are not present, then the reset value is 0x0000000X. If the instruction cache and data cache are present, then the reset value is 0x0000300X.	5.13 MSCR, Memory System Control Register on page 84
0xE001E004	PFCR	RW	0x00000001	5.15 PFCR, Prefetcher Control Register on page 88
0xE001E010	ITCMCR	RW	0x000000XX	5.19 ITCMCR and DTCMCR, TCM Control Registers on page 97
0xE001E014	DTCMCR	RW	0x000000XX	
0xE001E018	PAHBCR	RW	0x0000000X	5.14 PAHBCR, P-AHB Control Register on page 86
0xE001E100	IEBR0	RW	0x00000000	5.12.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1 on page 79
0xE001E104	IEBR1	RW	0x00000000	
0xE001E110	DEBR0	RW	0x00000000	5.12.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1 on page 80
0xE001E114	DEBR1	RW	0x00000000	
0xE001E120	TEBR0	RW	0x00000000	5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1 on page 82

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
-	TEBRDATA0	Not accessible from software	-	5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1 on page 83
0xE001E128	TEBR1	RW	0x00000000	5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1 on page 82
-	TEBRDATA1	Not accessible from software	-	5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1 on page 83
0xE001E200	DCADCRR	RO	**UNKNOWN**	5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74
0xE001E204	DCAICRR	RO	**UNKNOWN**	
0xE001E210	DCADCLR	RW	0x00000000	5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers on page 72
0xE001E214	DCAICLR	RW	0x00000000	
0xE001E300	CPDLPSTATE	RW	0x00000333	5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register on page 89
0xE001E304	DPDLPSTATE	RW	0x00000003	5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register on page 90
0xE001E400	EVENTSPR	WO	0x0000000X	5.21.1 EVENTSPR, Event Set Pending Register on page 104
0xE001E480	EVENTMASKA	RO	0x0000000X	5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers on page 105
0xE001E484+4n	EVENTMASKn	RO	**UNKNOWN**	
0xE001E500	ITGU_CTRL	RW	0x00000003	5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers on page 99
0xE001E504	ITGU_CFG	RO	0xX0002X0X	5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100
0xE001E510+4n	ITGU_LUTn	• RW if $32n+1<2^{\text{Number of ITGU blocks}}$ • RO if $32n+1\geq 2^{\text{Number of ITGU blocks}}$	0x00000000	5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers on page 101
0xE001E600	DTGU_CTRL	RW	0x00000003	5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers on page 99
0xE001E604	DTGU_CFG	RO	0xX0002X0X	5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE001E610+4n	DTGU_LUTn	• RW if $32n+1<2^{\text{Number of DTGU blocks}}$ • RO if $32n+1\geq2^{\text{Number of DTGU blocks}}$	0x00000000	5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers on page 101
0xE001E700	CFGINFOSEL	WO	UNKNOWN	5.17.1 CFGINFOSEL, Processor configuration information selection register on page 92
0xE001E704	CFGINFORD	RO	UNKNOWN	5.17.2 CFGINFORD, Processor configuration information read data register on page 95

Note

The following registers are reset on Cold reset only. These reset values persist across a system reset or Warm reset.

- 11.4.1 ERRFR0, RAS Error Record Feature Register on page 208.
- 11.4.4 ERRMISC10, Error Record Miscellaneous Register 10 on page 213.
- 11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers on page 212.
- 11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register on page 209.
- 11.4.5 ERRGSR0, RAS Fault Group Status Register on page 214.

5.11 Direct cache access registers

The Cortex®-M85 processor provides a set of **IMPLEMENTATION DEFINED** registers that allows direct read access to the embedded RAM associated with the L1 instruction and data cache. Two registers are included for each cache, one to set the required RAM and location, and the other to read out the data.

The following table lists the direct cache access registers.

Table 5-15: Direct cache access registers

Address	Name	Type	Reset value	Description
0xE001E200	DCADCRR	RO	UNKNOWN	5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74
0xE001E204	DCAICRR	RO	UNKNOWN	
0xE001E210	DCADCLR	RW	0x00000000	5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers on page 72
0xE001E214	DCAICLR	RW	0x00000000	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers

The DCAICLR and DCADCLR registers are used by software to set the location to be read from the L1 instruction cache and data cache respectively.

Usage Constraints

The DCAICLR is RAZ/WI if the L1 instruction cache is not present. The DCADCLR is RAZ/WI if the L1 data cache is not present. These registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

These registers are always implemented.

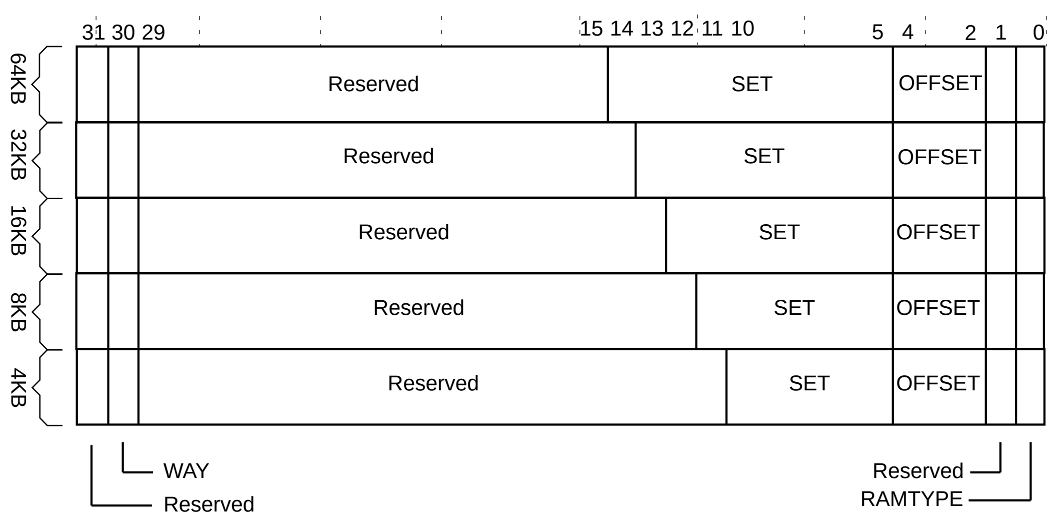
Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

DCAICLR

The following figure shows the DCAICLR bit assignments.

Figure 5-9: DCAICLR bit assignments



The following table shows the DCAICLR bit assignments.

Table 5-16: DCAICLR bit assignments

Bits	Name	Type	Function
[31]	Reserved	-	RES0
[30]	WAY	RO	Cache way

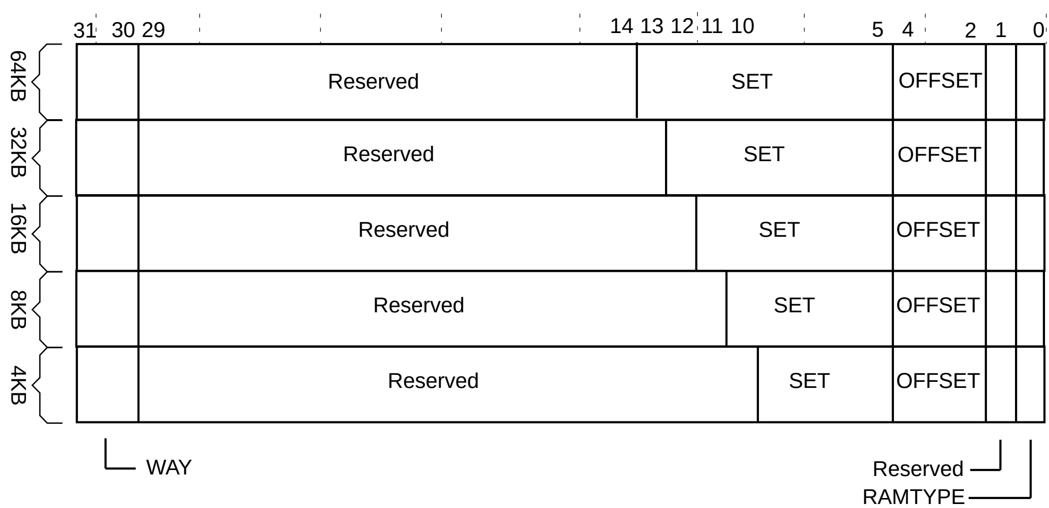
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Function
[29:N+1]	Reserved	-	
[N:5]	SET	RO	Set index. The value of N depends on the cache size. The options are: **64KB** N=14 **32KB** N=13 **16KB** N=12 **8KB** N=11 **4KB** N=10
[4:2]	OFFSET	RO	Data offset
[1]	Reserved	-	**RES0**
[0]	RAMTYPE	RO	RAM type **0** Tag RAM **1** Data RAM

DCADCLR

The following figure shows the DCADCLR bit assignments.

Figure 5-10: DCADCLR bit assignments



The following table shows the DCADCLR bit assignments.

Table 5-17: DCADCLR bit assignments

Bits	Name	Type	Function
[31:30]	WAY	RO	Cache way

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Function
[29:N+1]	Reserved	-	
[N:5]	SET	RO	Set index. The value of N depends on the cache size. The options are: **64KB** N=13 **32KB** N=12 **16KB** N=11 **8KB** N=10 **4KB** N=9
[4:2]	OFFSET	RO	Data offset
[1]	Reserved	-	**RES0**
[0]	RAMTYPE	RO	RAM type **0** Tag RAM **1** Data RAM

5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers

The *Direct Cache Access Instruction Cache Read Register* (DCAICRR) and *Direct Cache Access Data Cache Read Register* (DCADCRR) registers are used by software to read the data from the L1 instruction cache and data cache from the location that the DCAICLR and DCADCLR registers determine.

Usage Constraints

The DCAICRR is RAZ if the L1 instruction cache is not present. The DCADCRR is RAZ if the L1 data cache is not present.

This register is RAZ from the Non-secure state. Unprivileged access results in a BusFault exception.

These registers are also RAZ/WI if any of the following conditions are true:

- MSCR.ICACTIVE or MSCR.DCACTIVE is 0.
- PDRAMS is not powered up and clocked.
- The instruction or data cache is being automatically invalidated.

Configurations

These registers are always implemented.

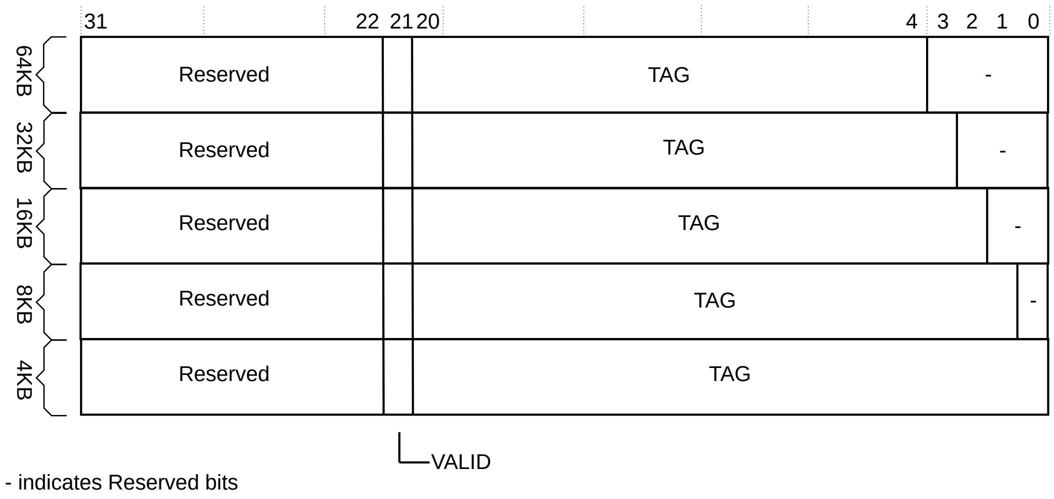
Attributes

These registers are read-only and ignore all writes. These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the DCAICRR bit assignments when reading the instruction cache tag RAM.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-11: DCAICRR bit assignments when reading the instruction cache tag RAM



The following table shows the DCAICRR bit assignments when reading the instruction cache tag RAM.

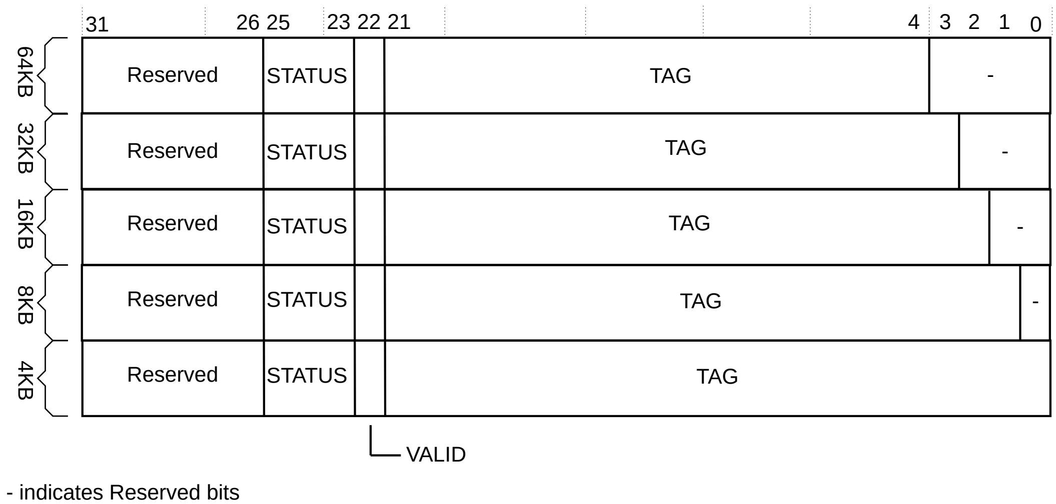
Table 5-18: DCAICRR bit assignments when reading the instruction cache tag RAM

Bits	Name	Type	Function
[31:22]	-	-	RES0
[21]	VALID	RO	Valid state of the instruction cache line.
[20:N]	TAG	RO	Tag address. The number of significant bits of TAG depends on the instruction cache size. **64KB** N=4 **32KB** N=3 **16KB** N=2 **8KB** N=1 **4KB** N=0
[N-1:0]	-	-	RES0, when N is not 0.

The following figure shows the DCADCRR bit assignments when reading the data cache tag RAM.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-12: DCADCRR bit assignments when reading the data cache tag RAM



The following table shows the DCADCRR bit assignments when reading the data cache tag RAM.

Table 5-19: DCADCRR bit assignments when reading the data cache tag RAM

Bits	Name	Type	Function
[31:26]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Function

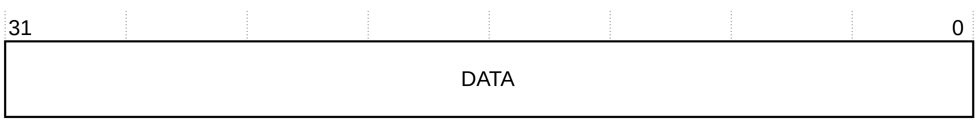
[25:23]	STATUS	RO	Clean or dirty, transient, and outer attributes of the cache line. The attribute encoding is as follows:
			`0b000`
			• Cache line is clean.
			• Cache line is transi
			• Outer attribu of the cache line are UNKN
			`0b001`
			• Cache line is clean.
			• Cache line is not transi
			• Outer attribu of the cache line are UNKN
			`0b010`
			• Cache line is dirty.
			• Cache line is not transi
			• Outer attribu of the cache line are Non-cache
			`0b011`
			• Cache line is dirty.
			• Cache line is not transi
			• Outer attribu

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Function
[22]	VALID	RO	Valid state of the data cache line entry.
[21:N]	TAG	RO	Tag address. The number of significant bits of TAG depends on the data cache size. **64KB** N=4 **32KB** N=3 **16KB** N=2 **8kB** N=1 **4KB** N=0
[N-1:0]	-	-	**RES0**, when N is not 0.

The following figure shows the DCAICRR and DCADCRR bit assignments when reading the instruction or data cache data RAM.

Figure 5-13: DCAICRR and DCADCRR bit assignments when reading the instruction or data cache data RAM



The following table shows the DCAICRR and DCADCRR bit assignments when reading the instruction or data cache data RAM.

Table 5-20: DCAICRR and DCADCRR bit assignments when reading the instruction or data cache data RAM

Bits	Name	Type	Function
[31:0]	DATA	RO	Instruction or data cache data entry, ignoring *Error Correcting Code* (ECC).

5.12 Error bank registers

When the Cortex®-M85 processor is configured to support *Error Correcting Code* (ECC) logic, these registers record errors which occur during memory accesses to the L1 instruction and data cache and the TCM. They also allow certain memory locations to be locked so hard errors can be contained and corrected.

The following table lists the error bank registers.

Table 5-21: Error bank registers

Address	Name	Type	Reset value	Description
`0xE001E100`	IEBR0	RW	`0x00000000`	5.12.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1 on page 79
`0xE001E104`	IEBR1	RW	`0x00000000`	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE001E110`	DEBR0	RW	`0x00000000`	5.12.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1 on page 80
`0xE001E114`	DEBR1	RW	`0x00000000`	
`0xE001E120`	TEBR0	RW	`0x00000000`	5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1 on page 82
-	TEBRDATA0	Not accessible from software	-	5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1 on page 83
`0xE001E128`	TEBR1	RW	`0x00000000`	5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1 on page 82
-	TEBRDATA1	Not accessible from software	-	5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1 on page 83

5.12.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1

The IEBR0 and IEBR1 registers are the two error bank registers that are included for the L1 instruction cache. These registers are used to record errors that occur during memory accesses to the L1 instruction cache. They also allow certain memory locations to be locked so hard errors can be contained and corrected.

Usage Constraints

These registers are not banked between security states. If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state, and are only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

Configurations

These registers are RAZ/WI if the L1 instruction cache is not present or if *Error Correcting Code* (ECC) is excluded.

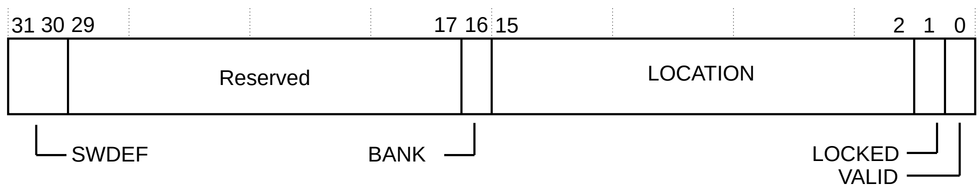
Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the IEBR0 and IEBR1 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-14: IEBR0 and IEBR1 bit assignments



The following table shows the IEBR0 and IEBR1 bit assignments.

Table 5-22: IEBR0 and IEBR1 bit assignments

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to `0b00` on a new allocation and on Cold reset.
[29:17]	Reserved	-	**RES0**
[16]	BANK	RW	Indicates which RAM bank to use. **0** Tag RAM. **1** Data RAM.
[15:2]	LOCATION	RW	Indicates the location in the L1 instruction cache RAM. **[15]** Way **[14:5]** Index **[4:3]** Line doubleword offset. **[2]** Reserved.
[1]	LOCKED	RW	Indicates whether the location is locked or not. **0** Location is not locked and available for hardware to allocate. **1** Software has locked the location and hardware is not allowed to allocate to this entry. Only one IEBRn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.
[0]	VALID	RW	Indicates whether the entry is valid or not. **0** Entry is invalid. **1** Entry is valid. The Cold reset value is 0.

5.12.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1

The DEBR0 and DEBR1 registers are the two error bank registers that are included for the L1 data cache. These registers are used to record errors that occur during memory accesses to the L1 data

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

cache. They also allow certain memory locations to be locked so hard errors can be contained and corrected.

Usage Constraints

These registers are not banked between security states. If and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state, and are only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

Configurations

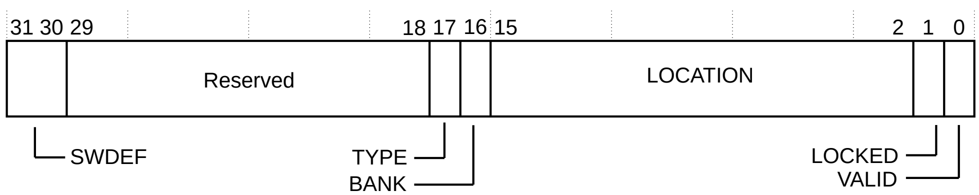
These registers are RAZ/WI if the L1 data cache is not present or if *Error Correcting Code* (ECC) is excluded.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the DEBR0 and DEBR1 bit assignments.

Figure 5-15: DEBR0 and DEBR1 bit assignments



The following table shows the DEBR0 and DEBR1 bit assignments.

Table 5-23: DEBR0 and DEBR1 bit assignments

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to `0b00` on a new allocation and on Cold reset.
[29:18]	Reserved	-	**RES0**
[17]	TYPE	RW	Indicates the error type. **0** Single-bit error. **1** Multi-bit error.
[16]	BANK	-	Indicates which RAM bank to use. **0** Tag RAM. **1** Data RAM.
[15:2]	LOCATION	-	Indicates the location in the data cache RAM. **[15:14]** Way. **[13:5]** Index. **[4:3]** Line doubleword offset. **[2]** Reserved.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Function
[1]	LOCKED	RW	Indicates whether the location is locked or not. **0** Location is not locked and available for hardware to allocate. **1** Software has locked the location and hardware is not allowed to allocate to this entry. Only one DEBRn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.
[0]	VALID	RW	Indicates whether the entry is valid or not. **0** Entry is invalid. **1** Entry is valid. The Cold reset value is 0.

5.12.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1

The TEBR0 and TEBR1 registers record the location of errors in the TCM.

Usage Constraints

These registers are not banked between security states. If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state, and are only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

Configurations

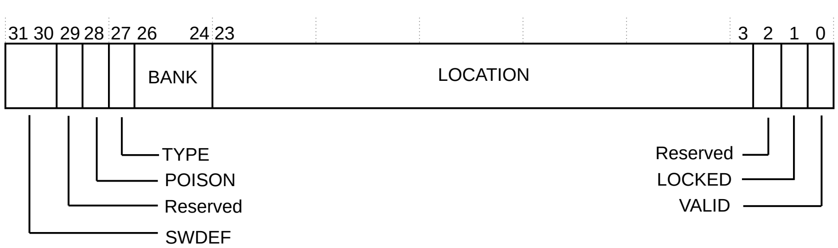
If *Error Correcting Code* (ECC) is excluded, these registers are RAZ/WI.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the TEBR0 and TEBR1 bit assignments.

Figure 5-16: TEBR0 and TEBR1 bit assignments



The following table shows the TEBR0 and TEBR1 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 5-24: TEBR0 and TEBR1 bit assignments

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to `0b00` on a new allocation and on Cold reset.
[29]	Reserved	-	**RES0**
[28]	POISON	RW	Indicates whether a BusFault is generated or not. **0** Load or non-word store (RMW) to an address that hits this TEBR accesses the corresponding TEBRDATA register and does not get a BusFault. **1** Load to address that hits this TEBR gets a BusFault. Non-word store (RMW) to an address that hits this TEBR aborts the write.
[27]	TYPE	RW	Indicates the error type. **0** Single-bit error. **1** Multi-bit error.
[26:24]	BANK	RW	Indicates which RAM bank to use. **`0b000`** DTCM0 **`0b001`** DTCM1 **`0b010`** DTCM2 **`0b011`** DTCM3 **`0b100`** ITCM All other values are **RES0**.
[23:3]	LOCATION	RW	Indicates the physical location in the data cache RAM.
[2]	Reserved	-	**RES0**
[1]	LOCKED	RW	Indicates whether the location is locked or not. **0** Location is not locked and available for hardware to allocate. **1** Software has locked the location and hardware is not allowed to allocate to this entry. Only one TEBRn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.
[0]	VALID	RW	Indicates whether the entry is valid or not. **0** Entry is invalid. **1** Entry is valid. If software programs both TEBRn registers with the same LOCATION and BANK field values and VALID is set to 1, then the behavior of TCM accesses is **UNPREDICTABLE**. The Cold reset value is 0.

5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1

The TEBRDATA0 and TEBRDATA1 registers provide storage for corrected data that is associated with an error.

Usage Constraints

These registers are not accessible from software.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Configurations

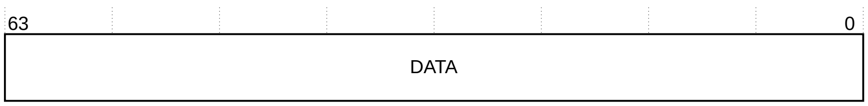
These registers are always implemented.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the TEBRDATA0 and TEBRDATA1 bit assignments.

Figure 5-17: TEBRDATA0 and TEBRDATA1 bit assignments



The following table shows the TEBRDATA0 and TEBRDATA1 bit assignments.

Table 5-25: TEBRDATA0 and TEBRDATA1 bit assignments

Bits	Name	Type	Function
[63:0]	DATA	-	These registers are not accessible from software.

5.13 MSCR, Memory System Control Register

The MSCR controls the memory system features specific to the Cortex®-M85 processor.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

Configuration

This register is always implemented and is read-only when the data cache is not included.

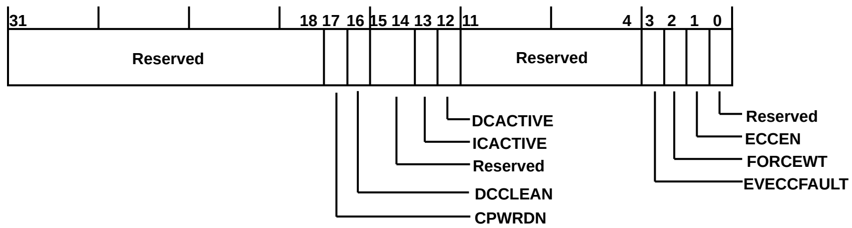
Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the MSCR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-18: MSCR bit assignments



The following table describes the MSCR bit assignments.

Table 5-26: MSCR bit assignments

Bits	Name	Type	Description
[31:18]	Reserved	-	**RES0**
[17]	CPWRDN	RO	This bit indicates when the data and instruction caches are not accessible because they are either being powered down or being initialized using the automatic invalidation sequence. Software that is enabling the cache can use this bit to determine when the cache is available for use. **0** Data and instruction cache in normal operational state. **1** Data and instructions cache powered down or automatic invalidation sequence is in process.
[16]	DCCLEAN	RW	This bit indicates whether the data cache contains any dirty lines. The options are: **0** L1 data cache contains at least one dirty line. **1** L1 data cache does not contain any dirty lines. It is cleared to 1 on any write to the L1 data cache that sets the dirty bit. It is cleared to 1 at the end of any automatic L1 data cache invalidate all. Software must only modify this register if it is restoring the state from before the core entered powerdown with the L1 data cache in retention. This field is not updated when a dirty line is evicted, therefore, MCSR.DCCLEAN can be 0, if the cache is currently clean but previously contained dirty data since the last time it was automatically invalidated. The reset value is 0. If the data cache is not included, this field is RAZ/WI.
[15:14]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bits	Name	Type	Description
[13]	ICACTIVE	RW	This bit indicates whether the L1 instruction cache is active. The options are: **0** L1 instruction cache is inactive. There is no allocation or lookups. Cache maintenance and direct cache access operations are treated as NOPs. **1** L1 instruction cache is active. This implies normal behavior. The reset value is 1. If the L1 instruction cache is not included, this field is RAZ/WI.
[12]	DCACTIVE	RW	This bit indicates whether the L1 data cache is active. The options are: **0** L1 data cache is inactive. There is no allocation or lookups. Cache maintenance and direct cache access operations are treated as NOPs. **1** L1 data cache is active. This implies normal behavior. The reset value is 1. If the L1 data cache is not included, this field is RAZ/WI.
[11:4]	Reserved	-	**RES0**
[3]	EVECCFAULT	RW	Enables asynchronous BusFault exceptions when data is lost on evictions. The options are: **0** Asynchronous BusFaults are not generated when evicting lines with multi-bit errors in the data. **1** Asynchronous aborts are generated when evicting lines with multi-errors in the data. This is intended for use in systems that do not support the AXI **xPOISON** signals. The reset value is 1. If ECC is not included, this field is RAZ/WI.
[2]	FORCEWT	RW	Enables Forced Write-Through in the L1 data cache. The options are: **0** Force Write-Through is disabled. **1** Force Write-Through is enabled. All Cacheable memory regions are treated as Write-Through. The reset value is 0. If the L1 data cache is not included, this field is RAZ/WI.
[1]	ECCEN	RO	Indicates whether *Error Correcting Code* (ECC) is present and enabled. The options are: **0** ECC not present or not enabled. **1** ECC present and enabled. The reset value depends on the `ECC` Verilog parameter and the external input signal **INITECCEN**. For more information on `ECC` Verilog parameter, see the *RTL configuration* section in the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. If ECC is not included, this field is RAZ/WI.
[0]	Reserved	-	**RES0**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.14 PAHBCR, P-AHB Control Register

The PAHBCR enables accesses to *Peripheral AHB* (P-AHB) interface from software running on the processor. This register also provides information on the range of memory-mapped to the interface.

The P-AHB is always memory-mapped to a range of the Peripheral and Vendor_SYS regions of the memory map. For more information on the memory map, see 8.1 Memory map on page 132.

Usage Constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

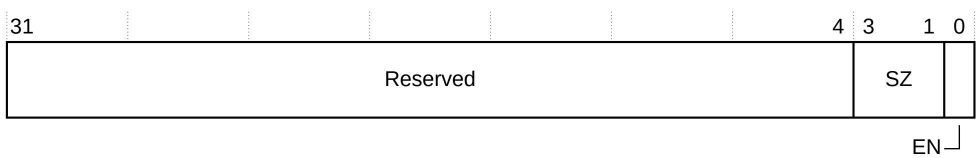
Configuration

This register is always implemented.

Attributes

See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

Figure 5-19: PAHBCR bit assignments



The following table shows the PAHBCR bit assignments.

Table 5-27: PAHBCR bit assignments

Bits	Name	Type	Description
[31:4]	-	-	Reserved.
[3:1]	SZ	RO	P-AHB size. The options are: **`0b000`** 0MB. This implies that P-AHB disabled. **`0b001`** 64MB. **`0b010`** 128MB. **`0b011`** 256MB. **`0b100`** 512MB. Other encodings are reserved. At reset, the register field is loaded from the **CFGPAHBSZ** input signal. The **CFGPAHBSZ** signal determines the size of the peripheral port memory region.
[0]	EN	RW	P-AHB enable. The options are: **0** P-AHB disabled. When disabled all accesses are made to the M-AXI interface. **1** P-AHB enabled. The reset value is derived from the **INITPAHBEN** signal. This field only affects accesses in the Peripheral region of the memory map. Accesses from the Vendor_SYS region are always enabled.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.15 PFCR, Prefetcher Control Register

The PFCR controls the Data cache prefetcher. This register can be used to enable or disable prefectching and to tune the prefetcher to optimize performance.

Usage Constraints

If AIRCR.BFHFNMINS is 0, then this register is RAZ/WI from Non-secure state. Unprivileged access causes a BusFault exception.

Configuration

This register is always implemented and is RAZ/WI when the L1 data cache is not included.

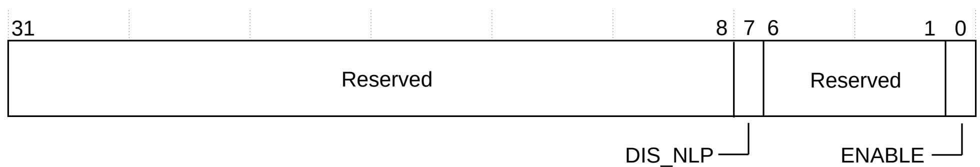
Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

For more information about data prefetching and prefetching modes, see 10.4.1.2 Data prefetching on page 165.

The following figure shows the PFCR bit assignments.

Figure 5-20: PFCR bit assignments



The following table shows the PFCR bit assignments.

Table 5-28: PFCR bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	**RES0**
[7]	DIS_NLP	RW	Disables Next Line Prefetch mode. The options are: **0** Next Line Prefetch is enabled. **1** Next Line Prefetch is disabled.
[6:1]	Reserved	-	**RES0**
[0]	ENABLE	RW	Prefetcher enable. The options are: **0** Prefetcher is disabled. **1** Prefetcher is enabled. The reset value is `0b1`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.16 Power mode control registers

The CPDLPSTATE and DPDLPSTATE registers allow software to control the required power mode of the functional and debug logic in the Cortex®-M85 processor.

The following table lists the power mode control registers.

Table 5-29: Power mode control registers

Address	Name	Type	Reset value	Description
`0xE001E300`	CPDLPSTATE	RW	`0x00000XX3` **Note:** Bits [9:8] and [5:4] can be RAZ/WI depending on your processor implementation. See 5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register on page 89 for more information.	5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register on page 89
`0xE001E304`	DPDLPSTATE	RW	`0x00000003`	5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register on page 90

5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register

The CPDLPSTATE register specifies the required low-power states for core (PDCORE), *Extension Processing Unit* (PDEPU), and RAM (PDRAMS) power domains.

Usage Constraints

If AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

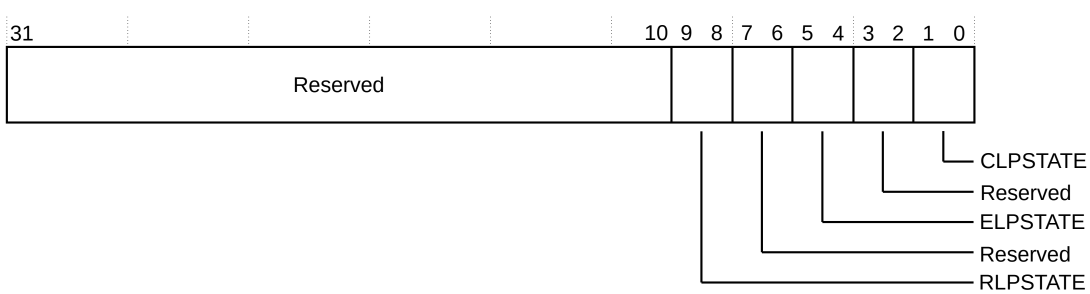
This register is always implemented.

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the CPDLPSTATE bit assignments.

Figure 5-21: CPDLPSTATE bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table shows the CPDLPSTATE bit assignments.

Table 5-30: CPDLPSTATE bit assignments

Bits	Name	Type	Function
[31:10]	Reserved	-	RES0
[9:8]	RLPSTATE	RW	Powerup state for PDRAMS power domain. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. The actual transition of the power mode is performed by the P-Channel. `0b00` ON `0b01` Reserved `0b10` Reserved `0b11` OFF **Note:** This field is used only to control the Cache/No cache operating mode for the P-Channel. RAM retention is enabled by entering any of the following power modes: • MEM_RET (Cache) • FULL_RET (Cache) • LOGIC_RET (Cache) For more information, 7.4 Core P-Channel and power mode selection on page 121. If the L1 data cache and instruction cache are not present, this field is RAZ/WI. The reset value is `0b11` on Cold reset.
[7:6]	Reserved	-	RES0
[5:4]	ELPSTATE	RW	Type of low-power state for PDEPU. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. The actual transition of the power mode is performed by the P-Channel. `0b00` ON. PDEPU is not in low-power state `0b01` ON, but the clock is off `0b10` RET `0b11` OFF If the *Extension Processing Unit* (EPU) is not present, this field is RAZ/WI. The reset value is `0b11` on Cold reset.
[3:2]	Reserved	-	RES0
[1:0]	CLPSTATE	RW	Type of low-power state for PDCORE. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. The actual transition of the power mode is performed by the P-Channel. `0b00` ON. PDCORE is not in low-power state `0b01` ON, but the clock is off `0b10` RET `0b11` OFF The reset value is `0b11` on Cold reset.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register

The DPDLPSTATE register specifies the required low-power states for the debug (PDDEBUG) power domain.

Usage Constraints

If AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

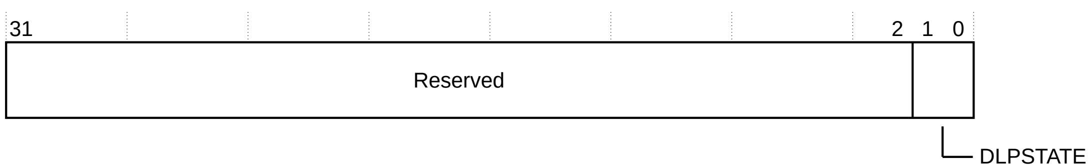
This register is always implemented.

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the DPDLPSTATE bit assignments.

Figure 5-22: DPDLPSTATE bit assignments



The following table shows the DPDLPSTATE bit assignments.

Table 5-31: DPDLPSTATE bit assignments

Bits	Name	Type	Function
[31:2]	Reserved	-	RES0
[1:0]	DLPSTATE	RW	Type of low-power state for PDDEBUG. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. **`0b00`** ON. PDDEBUG is not in low-power state. **`0b01`** ON, but the clock is off. **`0b10`** RESERVED. Treated as ON, but clock OFF. **`0b11`** OFF. The reset value is `0b11` at debug Cold reset, which is controlled by the **nDBGRESET** signal.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.17 Processor configuration information registers

The CFGINFOSEL and CFGINFORD registers provide information about the configuration of the processor including the values of all the Verilog parameters used during synthesis and input wire tie-off signals.

See 3.6 Cortex-M85 implementation options on page 39 for more information on the processor configuration options. For more detail on the RTL parameter values, see the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document that is available to licensees only and Arm partners with an NDA agreement.

The following table lists the processor configuration information registers.

Table 5-32: Processor configuration information registers

Address	Name	Type	Reset value	Description
`0xE001E700`	CFGINFOSEL	WO	UNKNOWN	5.17.1 CFGINFOSEL, Processor configuration information selection register on page 92
`0xE001E704`	CFGINFORD	RO	UNKNOWN	5.17.2 CFGINFORD, Processor configuration information read data register on page 95

5.17.1 CFGINFOSEL, Processor configuration information selection register

The CFGINFOSEL register selects the configuration information which can then be read back using CFGINFORD.

Usage constraints

Unprivileged access results in a BusFault exception.

This register is accessible through unprivileged Debug AHB (D-AHB) debug requests when either **DAUTHCTRL_S.UIDAPEN** or **DAUTHCTRL_NS.UIDAPEN** is set.

Configurations

This register is always implemented.

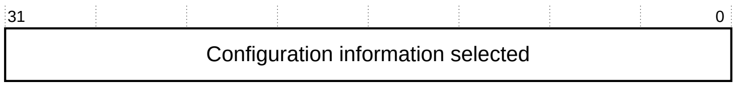
Attributes

This register is banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the CFGINFOSEL bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-23: CFGINFOSEL bit assignments



The following table describes the CFGINFOSEL bit assignments.

Table 5-33: CFGINFOSEL bit assignments

Field	Name	Type	Description
[31:0]	Configuration information selected	WO	The value of this field depends on the configuration information selected.

The following table lists the CFGINFOSEL register value that depends on the configuration information selected. For more information on the configuration parameters that are listed in the following table, see 3.6 Cortex-M85 implementation options on page 39.

Table 5-34: Configuration parameter selection used by the CFGINFOSEL register

CFGINFOSEL value	Configuration information selected
0x1	ICACHESZ
0x2	DCACHESZ
0x3	ECC
0x4	FPU
0x5	MVE
0x6	Reserved
0x7	CPIF
0x8	MPU_NS
0x9	MPU_S
0xA	SAU
0xB	ITGU
0xC	ITGUBLKSZ
0xD	ITGUMAXBLKS
0xE	DTGU
0xF	DTGUBLKSZ
0x10	DTGUMAXBLKS
0x11	NUMIRQ
0x12	IRQLVL
0x20+n, where 0≤ n≤ 0xF	Reserved
0x30+n, where 0≤ n≤ 0xF	IRQDIS[(n*32)+31:(n*32)]
0x40	Reserved
0x41	Reserved
0x42	DBGLVL
0x43	ITM

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

CFGINFOSEL value	Configuration information selected
0x44	ETM
0x45	Reserved
0x46	Reserved
0x47	IWIC
0x48	WICLINES
0x49	Reserved
0x4A	RAR
0x4B	INITL1RSTDIS
0x4C	CFGMEMALIAS
0x4D	Reserved
0x4E	Reserved
0x4F	PACBTI
0x50	Reserved
0x51	IDCACHEID

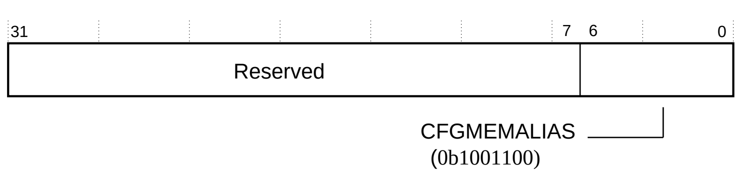
Note

- INITL1RSTDIS and CFGMEMALIAS select the corresponding external input wire tie-off signal value.
- Input wire tie-off signals also affect the FPU, MVE, MPU_NS, MPU_S, SAU and PACBTI values that are read. These signals are **CFGFPU**, **CFGMVE**, **MPUNSDISABLE**, **MPUSDISABLE**, **SAUDISABLE** and **CFGPACBTI**, respectively. If the input wire tie-off disables the feature, then the configuration indicates that the feature is not supported.
- The IRQDIS parameter is selected across multiple values.

CFGINFOSEL register value examples

The following figure shows the CFGINFOSEL bit assignments when CFGMEMALIAS parameter is selected.

Figure 5-24: CFGINFOSEL bit assignments showing CFGMEMALIAS



The following table describes the CFGINFOSEL bit assignments when CFGMEMALIAS parameter is selected.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 5-35: CFGINFOSEL bit assignments showing `CFGMEMALIAS`

Field	Name	Type	Description
[31:7]	Reserved	-	**RES0**
[6:0]	`CFGMEMALIAS`	WO	The value is `0x4C`.

5.17.2 CFGINFORD, Processor configuration information read data register

The CFGINFORD register can be used to display the configuration information that the CFGINFOSEL register selects.

Usage constraints

Unprivileged access results in a BusFault exception.

This register is accessible through unprivileged Debug AHB (D-AHB) debug requests when either **DAUTHCTRL_S.UIDAPEN** or **DAUTHCTRL_NS.UIDAPEN** is set.

Configurations

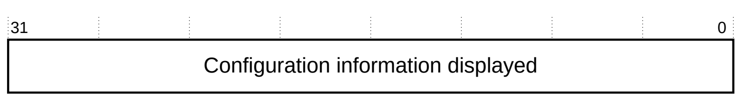
This register is always implemented.

Attributes

This register is banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the CFGINFORD bit assignments.

Figure 5-25: CFGINFORD bit assignments



The following table describes the CFGINFORD bit assignments.

Table 5-36: CFGINFORD bit assignments

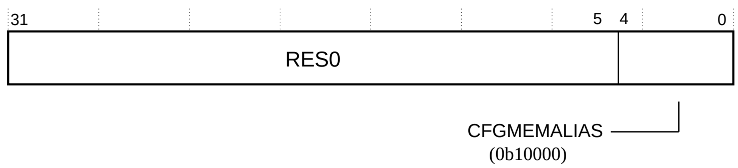
Field	Name	Type	Description
[31:0]	Configuration information displayed	RO	The value of this field depends on the configuration information selected.

CFGINFORD register value examples

The following figure shows the CFGINFORD bit assignments when the CFGINFOSEL register selects the `CFGMEMALIAS` parameter.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-26: CFGINFORD bit assignments showing `CFGMEMALIAS`



The following table describes the CFGINFORD bit assignments when **CFGMEMALIAS** configuration input signal is selected and the alias bit selected is 28.

Table 5-37: CFGINFORD bit assignments showing `CFGMEMALIAS`

Field	Name	Type	Description
[31:5]	Reserved	-	**RES0**
[4:0]	`CFGMEMALIAS`	RO	The value that is displayed is `0b10000` to indicate that alias bit 28 has been selected.

5.18 ID_PFR0, Processor Feature Register 0

The ID_PFR0 register contains a field that indicates the version of the *Reliability, Availability, and Serviceability* (RAS) extension supported.

Usage constraints

Unprivileged access results in a BusFault exception.

This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL_S.UIDAPEN or DAUTHCTRL_NS.UIDAPEN is set.

Configurations

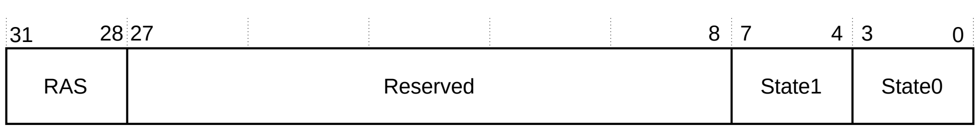
This register is always implemented.

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ID_PFR0 bit assignments.

Figure 5-27: ID_PFR0 bit assignments



The following table describes the ID_PFR0 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 5-38: ID_PFR0 bit assignments

Field	Name	Type	Description
[31:28]	RAS	RO	Identifies which version of the RAS architecture is implemented. `0b0010` Version 1.
[27:8]	Reserved	-	**RES0**
[7:4]	State1	RO	T32 instruction set support. `0b0011` T32 instruction set including Thumb-2 technology is implemented.
[3:0]	State0	RO	A32 instruction set support. `0b0000` A32 instruction set is not implemented.

5.19 ITCMCR and DTCMCR, TCM Control Registers

The ITCMCR and DTCMCR registers enable access to the *Tightly Coupled Memories* (TCMs) by software running on the processor. These registers also provide information on the physical size of the memory connected.

Usage Constraints

If AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

If the external input signal, **LOCKTCM** is asserted, these registers are read-only. For more information on **LOCKTCM**, see C.28 Miscellaneous signals on page 368.

Configuration

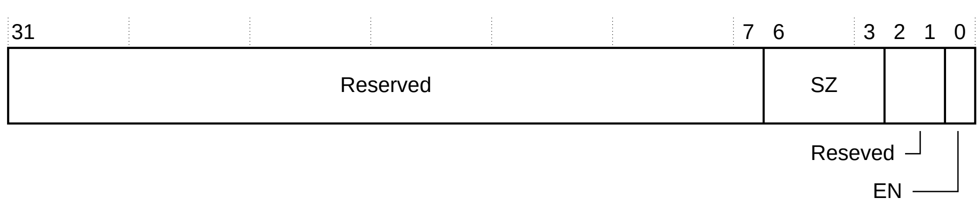
These registers are always implemented.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ITCMCR and DTCMCR bit assignments.

Figure 5-28: ITCMCR and DTCMCR bit assignments



The following table shows the ITCMCR and DTCMCR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 5-39: ITCMCR and DTCMCR bit assignments

Bits	Name	Type	Description
[31:7]	-	-	Reserved.
[6:3]	SZ	RO	TCM size indicates the size of the relevant TCM. The options are: `0b0000` No TCM implemented. `0b0011` 4KB `0b0100` 8KB `0b0101` 16KB `0b0110` 32KB `0b0111` 64KB `0b1000` 128KB `0b1001` 256KB `0b1010` 512KB `0b1011` 1MB `0b1100` 2MB `0b1101` 4MB `0b1110` 8MB `0b1111` 16MB All other encodings are reserved. The reset value is derived from the **CFGITCMSZ** and **CFGDTCMSZ** signals.
[2:1]	Reserved	-	RAZ/WI.
[0]	EN	RW	TCM enable. When a TCM is disabled all accesses are made to the *Master AXI* (M-AXI) interface. The options are: **0** TCM disabled. **1** TCM enabled. The reset value is derived from the **INITTCMEN** signal. This field only affects software accesses to the TCM. Accesses to the TCM from the *S-AHB* interface are always enabled.

5.20 TCM security gate registers

The TCM security gates that are associated with the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) are configured using the ITGU_CTRL and DTGU_CTRL registers, respectively. Additionally, there is a set of registers with a group of blocks, ITGU_LUTn and DTGU_LUTn. The configuration of a gate can be read from the read-only ITGU_CFG and DTGU_CFG registers.

The following table lists the TCM security gate registers.

Table 5-40: TCM security gate registers

Address	Name	Type	Reset value	Description
`0xE001E500`	ITGU_CTRL	RW	`0x00000003`	5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers on page 99

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE001E504`	ITGU_CFG	RO	`0xX0002X0X`	5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100
`0xE001E510+4n`	ITGU_LUTn	• RW if $32n+1<2^{\text{Number of ITGU blocks}}$ • RO if $32n+1\geq 2^{\text{Number of ITGU blocks}}$	`0x00000000`	5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers on page 101
`0xE001E600`	DTGU_CTRL	RW	`0x00000003`	5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers on page 99
`0xE001E604`	DTGU_CFG	RO	`0xX0002X0X`	5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100
`0xE001E610+4n`	DTGU_LUTn	• RW if $32n+1<2^{\text{Number of ITGU blocks}}$ • RO if $32n+1\geq 2^{\text{Number of ITGU blocks}}$	`0x00000000`	5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers on page 101

5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers

The ITGU_CTRL and DTGU_CTRL registers are the main *TCM Gate Unit* (TGU) control registers for the ITCM and DTCM respectively.

Usage constraints

These registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

If the external input signal **LOCKITGU** is asserted, the ITGU_CTRL register is read-only.

If the external input signal **LOCKDTGU** is asserted, the DTGU_CTRL register is read-only.

Configurations

These registers are always implemented, but their behavior depends on whether the ITGU and DTGU are present.

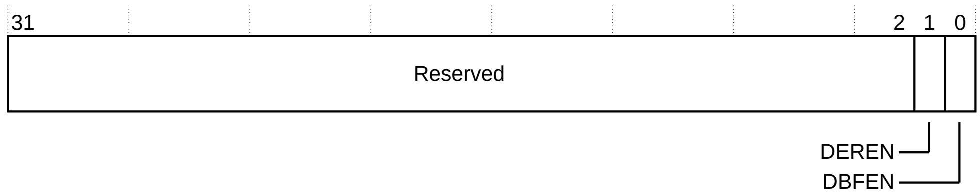
Attributes

These registers are not banked between Security states. For more information, see 5.10 IMPLEMENTATION DEFINED registers summary on page 68.

The following figure shows the ITGU_CTRL and DTGU_CTRL bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-29: ITGU_CTRL and DTGU_CTRL bit assignments



The following table describes the ITGU_CTRL and DTGU_CTRL bit assignments.

Table 5-41: ITGU_CTRL and DTGU_CTRL bit assignments

Field	Name	Type	Description
[31:2]	Reserved	-	-
[1]	DEREN	RW	Enable *Slave AHB* (S-AHB) error response for TGU fault. The options are: **0** Error response is not enabled. **1** Error response is enabled.
[0]	DBFEN	RW	Enable data side BusFault for TGU fault. The options are: **0** BusFault not enabled. **1** BusFault enabled.

5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers

The ITGU_CFG and DTGU_CFG registers allow the reading of configuration values for the ITGU and DTGU respectively.

Usage constraints

These registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

These registers are always implemented, but their behavior depends on whether the ITGU and DTGU are present.

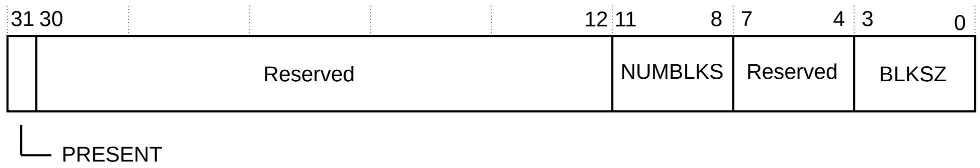
Attributes

These registers are not banked between Security states. For more information, see 5.10 IMPLEMENTATION DEFINED registers summary on page 68.

The following figure shows the ITGU_CFG and DTGU_CFG bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-30: ITGU_CFG and DTGU_CFG bit assignments



The following table describes the ITGU_CFG and DTGU_CFG bit assignments.

Table 5-42: ITGU_CFG and DTGU_CFG bit assignments

Field	Name	Type	Description
[31]	PRESENT	-	This field determines if the TGU is present. The options are: **0** TGU not present. **1** TGU is present
[30:12]	Reserved	-	**RES0**
[11:8]	NUMBLKS	RO	NUMBLKS=**CFGxTCMSZ**+4 -`xTGUBLKSZ` The number of TCM blocks is $2^{NUMBLKS}$. Where: • **CFGxTCMSZ** is the configured TCM size. • `xTGUBLKSZ` is the configured *Instruction Tightly Coupled Memory Gate Unit* (ITGU) or *Data Tightly Coupled Memory Gate Unit* (DTGU) block size.
[7:4]	Reserved	-	**RES0**
[3:0]	BLKSZ	RO	TGU block size in bytes. This is $2^{BLKSZ+5}$. This field is determined by the Verilog parameter `xTGUBLKSZ`.

5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers

The ITGU_LUTn and DTGU_LUTn registers allows identifying the TGU blocks as Secure or Non-secure, where n is in the range 0-15.

Usage constraints

These registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

If the external input signal **LOCKITGU** is asserted, the ITGU_LUTn register is read-only.

If the external input signal **LOCKDTGU** is asserted, the DTGU_LUTn register is read-only.

Configurations

The number of programmable blocks depends on the processor configuration and the physical TCM size. This is calculated using the following formula, where x is I for ITGU and D for DTGU:

$$N = 2^{xTGU\_CFG.NUMBLKS}$$

Accesses to register fields associated with blocks above the programmable number are treated as RAZ/WI. For more information on the ITGU_CFG and DTGU_CFG registers and the NUMBLKS

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

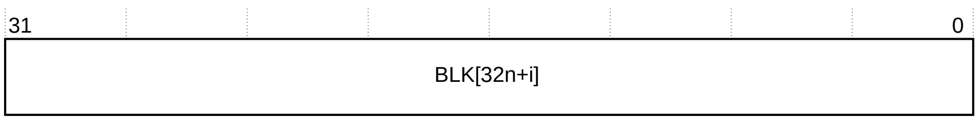
field, see 5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100 .

Attributes

These registers are not banked between Security states. For more information, see 5.10 IMPLEMENTATION DEFINED registers summary on page 68.

The following figure shows the ITGU_LUTn and DTGU_LUTn bit assignments.

Figure 5-31: ITGU_LUTn and DTGU_LUTn bit assignments



The following table describes the ITGU_LUTn and DTGU_LUTn bit assignments where:

- 0≤n≤15
- 0≤i≤31
- N is the number of programmable blocks: $N=2^{xTGU\_CFG.NUMBLKS}$
- x is I for ITGU and D for DTGU

Table 5-43: ITGU_LUTn and DTGU_LUTn bit assignments for implemented block mapping

Field	Name	Type	Description
[31:0]	BLK[32n+i]	• RW for 32n+i<N • RO for 32n+i≥N	If 32n+i<N, then the block 32n+i is implemented, and the security mapping bit options are: **0** Block mapped as Secure **1** Block mapped as Non-secure If 32n+i≥N, then the block 32n+i is not implemented, and the accesses are treated as RAZ/WI.

5.20.3.1 ITGU_LUTn and DTGU_LUTn example

Consider the following example to calculate ITGU_LUTn and DTGU_LUTn, with ITGU_CFG.NUMBLKS and DTGU_CFG.NUMBLKS set to 4.

Number of programmable blocks (N)=$2^{xTGU\_CFG.NUMBLKS}$

xTGU_CFG.NUMBLKS=**CFGxTCMSZ**+4-`xTGUBLKSZ`, where x can be I or D for ITCM and DTCM respectively.

If **CFGxTCMSZ** is `0b011` and `xTGUBLKSZ` is 3, then xTGU_CFG.NUMBLKS is 4.

$N=2^4$, that is 16.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Number of xTGU_LUTn registers

Up to 16 xTGU_LUTn registers can be configured which each register supporting 32 blocks, with n in the range 0-15. In this example, only one xTGU_LUT register is required, that is, ITGU_LUT and DTGU_LUT, where n=0.

Calculating the BLK[32n+i], where i is the bit offset in the register and can be in the range 0-31

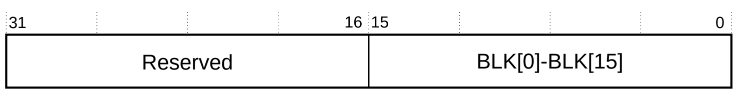
Since n=0 because all programmable blocks can fit into one 32-bit register, BLK is calculated as:

BLK[(32×0)+0] to BLK[(32×0)+15]. That is, BLK[0] to BLK[15].

Bit assignments

The following figure shows the bit assignments for xTGU_LUT when n=0.

Figure 5-32: ITGU_LUT and DTGU_LUT bit assignments



The following table describes the bit assignments.

Table 5-44: ITGU_LUTn and DTGU_LUTn bit assignments for implemented block mapping

Field	Name	Type	Description
[31:16]	-	RO	RAZ/WI.
[15:0]	BLK[0] to BLK[15]	RW	If 32n+i<N, then the implemented block 32n+i security mapping bit options are: **0** Block mapped as Secure. **1** Block mapped as Non-secure.

5.21 EWIC interrupt status access registers

The *External Wakeup Interrupt Controller* (EWIC) interrupt status access registers, EVENTSPR, EVENTMASKA, and EVENTMASKn registers provide access to the *Nested Vectored Interrupt Controller* (NVIC) state that must be used to carry out software transfers to and from the EWIC in the system for sleep entry and exit when the automatic transfer feature is disabled.

The following table lists the EWIC interrupt status access registers.

Table 5-45: EWIC interrupt status access registers

Address	Name	Type	Reset value	Description
`0xE001E400`	EVENTSPR	WO	UNKNOWN	5.21.1 EVENTSPR, Event Set Pending Register on page 104

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE001E480`	EVENTMASKA	RO	UNKNOWN	5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers on page 105
`0xE001E484+4n`	EVENTMASKn	RO	UNKNOWN	

5.21.1 EVENTSPR, Event Set Pending Register

The EVENTSPR is a write-only register that is used to set pending events at wakeup that cannot be directly set in the *Nested Vectored Interrupt Controller* (NVIC) using the architecture programming model.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

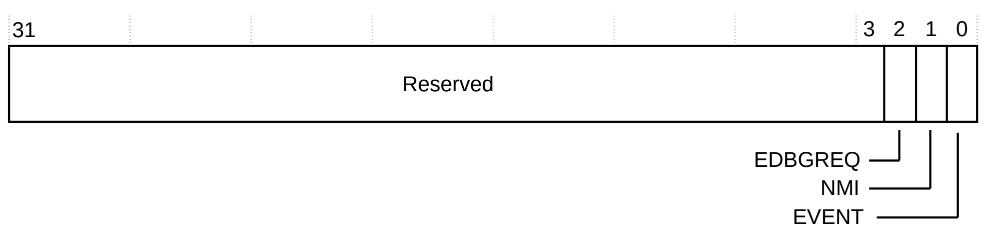
This register is always implemented.

Attributes

This register is not banked between Security states. For more information, see 5.10 IMPLEMENTATION DEFINED registers summary on page 68. The format of this register is identical to the EWIC_PEND0 register. For more information on the EWIC_PEND0 register, see A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers on page 306.

The following figure shows the EVENTPSR bit assignments.

Figure 5-33: EVENTSPR bit assignments



The following table describes the EVENTSPR bit assignments.

Table 5-46: EVENTSPR bit assignments

Field	Name	Type	Description
[31:3]	Reserved	-	RES0
[2]	EDBGREQ	WO	A write of one to this field causes the processor to behave as if an external debug request has occurred. A write of zero is ignored.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[1]	NMI	WO	A write of one to this field causes the processor to behave as if a non-maskable interrupt, NMI, has occurred. A write of zero is ignored.
[0]	EVENT	WO	A write of one to this field causes the processor to behave as if an RXEV event has occurred. A write of zero is ignored.

5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers

The EVENTMASKA and EVENTMASKn are read-only registers that provide the events on sleep entry which cause the processor to wake up. EVENTMASKA includes information about internal events and the EVENTMASKn registers cover external interrupt requests (IRQ). There is one register implemented for each of the 32 external interrupts that the *External Wakeup Interrupt Controller* (EWIC) supports. The EVENTMASKA register is always implemented.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

Configurations

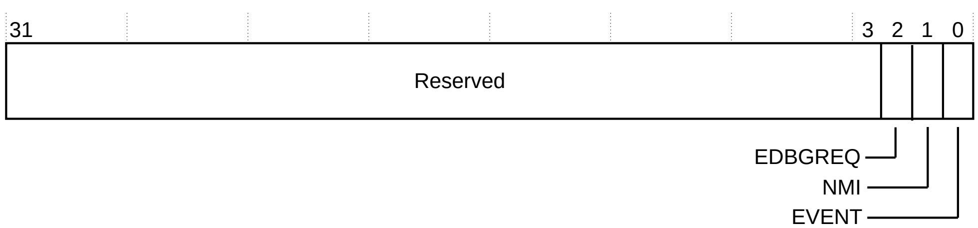
These registers are always implemented.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the EVENTMASKA bit assignments.

Figure 5-34: EVENTMASKA bit assignments



The following table describes the EVENTMASKA bit assignments.

Table 5-47: EVENTMASKA bit assignments

Field	Name	Type	Description
[31:3]	-	-	Reserved, **RES0**
[2]	EDBGREQ	RO	Mask for external debug request. If this bit is 0, the mask is enabled.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[1]	NMI	RO	Mask for NMI. If this bit is 0, the mask is enabled.[1]
[0]	EVENT	RO	Sensitive to RXEV when in WFE sleep

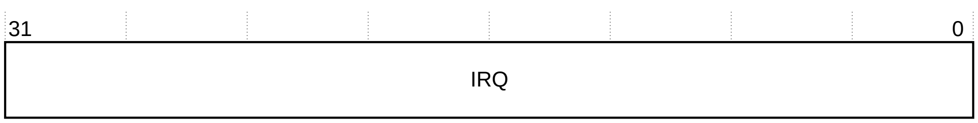
Note

EVENTMASKA[0] is **RES0** as the wakeup sensitivity to an external event is determined by the sleep entry instruction and not the processor state. The software transfer sequence must set the EWIC_MASKA.EVENT register field, if the sleep entry instruction is WFE.

EWIC_MASKA.EVENT should be set to `0b0` if the sleep entry instruction is not WFE. For more information on EWIC_MASKA, see A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers on page 305 .

The following figure shows the EVENTMASKn, where n=0-14, bit assignments.

Figure 5-35: EVENTMASKn, where 0≤n<15, bit assignments



The following table describes the EVENTMASKn, where n=0-14, bit assignments.

Table 5-48: EVENTMASKn, where 0≤n<15, bit assignments

Field	Name	Type	Description
[31:0]	IRQ	RO	Masks for interrupts (n×32) to ((n+1)×32)-1. If any of the bits are 0, the mask is enabled for the associated interrupt. Additionally, any interrupt that the WIC does not support is also RAZ.

[1] An NMI can be masked in certain cases where the execution priority is equal to or higher than NMI priority.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6 Initialization

This chapter describes how to initialize the Cortex®-M85 processor and which registers to access to enable functionality before using the processor features.

6.1 Initialization overview

Before your run your application, you might want to program values into registers and memory and enable certain processor features.

This chapter describes other initialization requirements, some of which are optional depending on the features you have implemented in the Cortex®-M85 processor.

6.2 Initializing and reprogramming the MPU

The Cortex®-M85 processor includes the *Memory Protection Unit* (MPU), which is primarily used for memory region protection.

Memory protection logic can be split between Secure and Non-secure MPU (MPU_S and MPU_NS).

The MPU_CTRL.ENABLE must be set to 1 to enable the MPU.

MPU_CTRL_NS is the Non-secure version of this register, and can be used to enable the Non-secure MPU region. For more information on MPU_CTRL, see the *Arm®v8-M Architecture Reference Manual*.

Note

For more information on the MPU, see 9.3 Memory Protection Unit on page 142.

Reprogramming the MPU

When setting up the MPU, and if it has been previously programmed, disable unused regions to prevent any old settings from affecting the latest MPU setup.

1. Execute a `DSB` instruction, to drain out any existing memory transactions.
2. Write to the MPU registers. For a complete list, see 9.3.1 Memory Protection Unit register summary on page 143.
3. Execute a `DSB` instruction and then an `ISB` instruction, to ensure that all subsequent memory accesses see the updated MPU setup.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Remember

Additionally, if any memory is converted from Cacheable to Non-cacheable or Device, and any write has been performed to that memory, you must perform data cache clean and invalidate operations (`DCIMVAC`) each of these cachelines.

For more information on these operations, see the *Arm®v8-M Architecture Reference Manual*.

6.3 Initializing the EPU

The *Extension Processing Unit* (EPU) is disabled on reset. The core must be in privileged mode to read from and write to the CPACR.

To allow the EPU to run Non-secure code, the NSACR must be setup by Secure privileged software.

The following code sequence demonstrates this:

```
NSACR EQU 0xE000ED8C
LDR R0, =NSACR ; Read NSACR
LDR r1, [R0] ; Set bits 10-11 to allow Non-secure access to CP10 and CP11
 coprocessors.
ORR R1, R1, #(0x3 << 10)
STR R1, [R0] ; Write back the modified value to the NSACR.
DSB
ISB ; Reset pipeline now the Non-secure access has been allowed to CP10 and CP11
 coprocessors.
```

To enable the EPU, privileged software must setup the CPACR, which is demonstrated by the following code sequence.

Note

The CPACR is banked between Security state and this code sequence enables the EPU for the current Security state only.

```
CPACR EQU 0xE000ED88
LDR R0, =CPACR ; Read CPACR
LDR r1, [R0] ; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
STR R1, [R0] ; Write back the modified value to the CPACR
DSB
ISB ; Reset pipeline now the EPU is enabled.
```

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.4 Programming the SAU

The *Security Attribution Unit* (SAU) is always present in the Cortex®-M85 processor and is used to define and authenticate accesses to memory.

At reset, before any SAU regions are programmed, the default internal security level is selected using the SAU_CTRL.ALLNS register. In the Cortex®-M85 processor, this register always resets to zero, setting most of the memory (except some regions in the PPB space) to Secure, and preventing an *Implementation Defined Attribution Unit* (IDAU) from overriding the security level.

However, after reset, Secure software can allow an IDAU to specify the security level for all memory regions by disabling all the SAU regions and setting SAU_CTRL.ALLNS to one.

To enable the SAU, Secure software must:

1. Program the regions that are required into the SAU_RBAR and SAU_RLAR registers. To change an SAU region, you must clean and invalidate any addresses from the previous configuration from the cache.
2. Set the SAU_CTRL.ENABLE bit to 1.

For more information on these registers, see *Arm®v8-M Architecture Reference Manual*.

The **LOCKSAU** signal prevents software accesses to the SAU registers. For more information on **LOCKSAU**, see C.28 Miscellaneous signals on page 368.

Note

For more information on the SAU and IDAU, see 9.2 Security Attribution Unit on page 140 and 9.4 Implementation Defined Attribution Unit on page 144

6.5 Initializing the instruction and data cache

On initial powerup, the instruction and data caches are in an **UNKNOWN** state. Therefore, on initial powerup, the caches must be initialized either by automatic invalidation or through software invalidation.

If you implement RAM retention without using the P-Channel, then software invalidation of caches might be required.

If a P-Channel is not used for RAM retention, you must do either of the following:

- Set **INITL1RSTDIS** to an appropriate value when the cache is valid on reset
- Tie **INITL1RSTDIS** HIGH and invalidate software.

The caches are not accessible during the automatic invalidation sequence. Executing a `DSB` instruction causes the processor to wait for the sequence to complete.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The CCR.DC and CCR.IC register bits are banked based on security, therefore each Security state must set these bits to enable the data and instruction cache.

For more information on the CCR register, see *Arm®v8-M Architecture Reference Manual*.

Note

You can optionally implement *Error Correcting Code* (ECC) functionality on caches by setting the `ECC` RTL parameter. However, the Cortex®-M85 processor does not support disabling ECC using software. Enabling and disabling ECC is done at Cold reset by the **INITECCEN** signal. For more information on **INITECCEN**, see C.4 Reset configuration signals on page 343.

For more information on instruction and data caches, see 10.9 Instruction and data cache on page 183.

6.5.1 Enabling the instruction and data cache

The following code sequence demonstrates how to enable the instruction and data cache for the current Security state when running in privileged mode.

```
CCR EQU 0xE000ED14
LDR R0, =CCR ; Read CCR
LDR r1, [R0] ; Set bits 16 and 17 to enable D-cache and I-cache
ORR R1, R1, #(0x3 << 16)
STR R1, [R0] ; Write back the modified value to the CCR
DSB
ISB ; Perform DSB and ISB to guarantee change is visible to subsequent instructions
```

6.5.2 Powering down the caches

To powerdown the caches:

1. Set CCR.DC and CCR.IC to 0. CPDLPSTATE.RLPSTATE must be set to `0b11`.
2. If the data cache contains dirty data that must be transferred to system memory, the entire cache must be cleaned with a set of Set/Way cache maintenance operations.

```
CCSIDR EQU 0xE000ED80 ; Current cache size ID register address
CSSELR EQU 0xE000ED84 ; Cache size selection register address
DCCSW  EQU 0xE000EF6C ; Cache maintenance op address: data cache clean by set/way
; CSSELR selects the cache visible in CCSIDR
MOV r0, #0x0 ; 0 = select "level 1 data cache"
LDR r11, =CSSELR ;
STR r0, [r11] ;
DSB ; Ensure write to CSSELR before proceeding
LDR r11, =CCSIDR ; From CCSIDR
LDR r2, [r11] ; Read data cache size information
AND r1, r2, #0x7 ; r1 = cache line size
ADD r7, r1, #0x4 ; r7 = number of words in a cache line
UBFX r4, r2, #3, #10 ; r4 = number of "ways"-1 of data cache
UBFX r2, r2, #13, #15 ; r2 = number of "set"-1 of data cache
CLZ r6, r4 ; calculate bit offset for "way" in DCISW
LDR r11, =DCCSW ; clean cache by set/way
inv_loop1 ; For each "set"
```

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
MOV r1, r4 ; r1 = number of "ways"-1
LSLS r8, r2, r7 ; shift "set" value to bit 5 of r8
inv_loop2 ; For each "way"
LSLS r3, r1, r6 ; shift "way" value to bit 30 in r6
ORRS r3, r3, r8 ; merge "way" and "set" value for DCISW
STR r3, [r11] ; invalidate D-cache line
SUBS r1, r1, #0x1 ; decrement "way"
BGE inv_loop2 ; End for each "way"
SUBS r2, r2, #0x1 ; Decrement "set"
BGE inv_loop1 ; End for each "set"
DSB ; Data sync barrier after invalidate cache
ISB ; Instruction sync barrier after invalidate cache
```

3. Set MSCR.DCACTIVE and MSCR.ICACTIVE to 0. As a result, the processor core deasserts bit 16 of the **COREPACTIVE** signal, which is a hint to the external power controller that PDRAMS can be powered down.

6.5.3 Powering up the caches

To powerup the caches:

1. Set MSCR.DCACTIVE and MSCR.ICACTIVE to 1. As a result, the processor core asserts **COREPACTIVE[16]**, to indicate to an external power controller that PDRAMS are required to be be powered up.
2. Set CCR.DC and CCR.IC to 1. After the external power control logic has powered up PDRAMS, the *Core Power Control* (CPC) triggers an automatic invalidation of the RAMs (if **INITL1RSTDIS** is 0), and after that is complete, subsequent instructions can cause allocations to and lookups in the caches.

6.6 Enabling branch prediction

Branch prediction is disabled on reset. Branch prediction improves performance by predicting the existence and targets of branch instructions in the early stages of the processor pipelines.

The processor core must be in privileged mode to read from and write to the CCR. The CCR.BP bit is banked so, it must be enabled for each Security state to enable branch prediction in that state. For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.

The following code sequence demonstrates how to enable branch prediction for the current Security state when running in privileged mode.

```
CCR EQU 0xE000ED14
LDR R0, =CCR ; Read CCR
LDR r1, [R0] ; Set bits 18 to enable branch prediction
ORR R1, R1, #(0x4 << 16)
STR R1, [R0] ; Write back the modified value to the CCR
DSB
ISB ; Reset pipeline now branch prediction is enabled.
```

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.7 Enabling the branch cache

The branch cache is disabled on reset. You must enable the branch cache to implement *Low Overhead Branch* (LOB) Extension.

The processor core must be in privileged mode to read from and write to the CCR. The CCR.LOB bit is banked so it must be enabled for each Security state that uses the LOB Extension. For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.

The following code sequence demonstrates how to enable the branch cache for the current Security state when running in privileged mode.

```
CCR EQU 0xE000ED14
LDR R0, =CCR ; Read CCR
LDR r1, [R0] ; Set bits 19 to enable LOB
ORR R1, R1, #(0x8 << 16)
STR R1, [R0] ; Write back the modified value to the CCR
DSB
ISB ; Reset pipeline now LOB is enabled.
```

6.8 Enabling and preloading the TCM

The Cortex®-M85 processor can optionally include *Tightly Coupled Memories* (TCMs).

Enabling the TCMs

For more information, see 10.8 TCM interfaces on page 178.

Software must set the ITCMCR.EN and DTCMCR.EN fields to enable access to the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) respectively. For more information on these registers, see 5.19 ITCMCR and DTCMCR, TCM Control Registers on page 97.

Alternatively, if the **INITTCMEN[1:0]** signal is asserted on Cold or Warm reset, then software does not need to write to these registers. For more information on the **INITTCMEN[1:0]** signal, see C.4 Reset configuration signals on page 343.

Preloading the TCMs

The methods to preload the TCMs are:

Memory copy with running boot code

When boot code includes a memory copy routine that reads data from a ROM and writes it into the appropriate TCM, you must enable the TCM to perform this operation. This bootcode must be run from an address outside the TCM region.

DMA into TCM

You can use a *Direct Memory Access* (DMA) device that reads data from a ROM and writes it to the TCMs through the *Slave AHB* (S-AHB) interface. This method can be used to preload the TCM so they can be used by the processor from reset.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Using the TCM from reset

If the TCM interface is configured to enable the TCMs at reset and the reset vector address is inside the TCM memory region, then the processor boots from TCM. The system must ensure that the bootcode software is present in the appropriate memory region before execution starts. This can be accomplished by either initializing the memory before reset or by transferring the data after reset using the S-AHB interface and asserting the **CPUWAIT** input signal. Asserting this signal stops the processor fetching or executing instructions after reset. When the **CPUWAIT** signal is deasserted the processor starts fetching instructions from the reset vector address in the normal way.

Note

Asserting **CPUWAIT** only takes effect when the processor is under processor reset or Cold reset, that is, **nSYSRESET** or **nPORESET** is asserted. The processor does not halt if **CPUWAIT** is asserted while the processor is running.

The ITCM and DTCM can be locked from software access using the external input signal, **LOCKTCM**. When this signal is asserted. it disables writes to registers that are associated with the TCM region from software or from a debug agent connected to the processor.

- ITCMCR.
- DTCMCR.

Asserting this signal prevents changes to the TCM configuration. All writes to the registers are ignored.

Note

When ECC is enabled, before performing a byte, halfword, or unaligned word write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned doubleword write to the location. Arm recommends that all TCM locations are initialized in this manner by boot code.

6.9 Enabling and locking the TCM security gates

TCM gating is enabled by tying the external input signal **CFGMEMALIAS** to a non-zero value.

The *TCM Gate Unit* (TGU) can be locked from software access using the external input signals **LOCKITGU** and **LOCKDTGU**. When these signals are asserted the corresponding TGU registers become read-only. This allows a TGU configuration to be programmed and then locked from further changes by software. For more information on TCM security gating, see 9.7 TCM Gate Units on page 146.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.10 Enabling the P-AHB interface

Software can enable the *Peripheral AHB* (P-AHB) interface by writing to the PAHBCR.EN register.

For more information on PAHBCR, see 5.14 PAHBCR, P-AHB Control Register on page 86.

Alternatively, you can assert **INITPAHBEN** HIGH at Cold or Warm reset, to enable the P-AHB interface. If you do this, there is no need for a software write to PAHBCR.EN. For more information on **INITPAHBEN**, see C.4 Reset configuration signals on page 343.

The P-AHB can be locked from software access using the external input signal, **LOCKPAHB**. When this signal is asserted, writes to PAHBCR register from software or from a debug agent connected to the processor are disabled and the register becomes read-only. Asserting this signal prevents changes to P-AHB port enable status in PAHBCR.EN.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

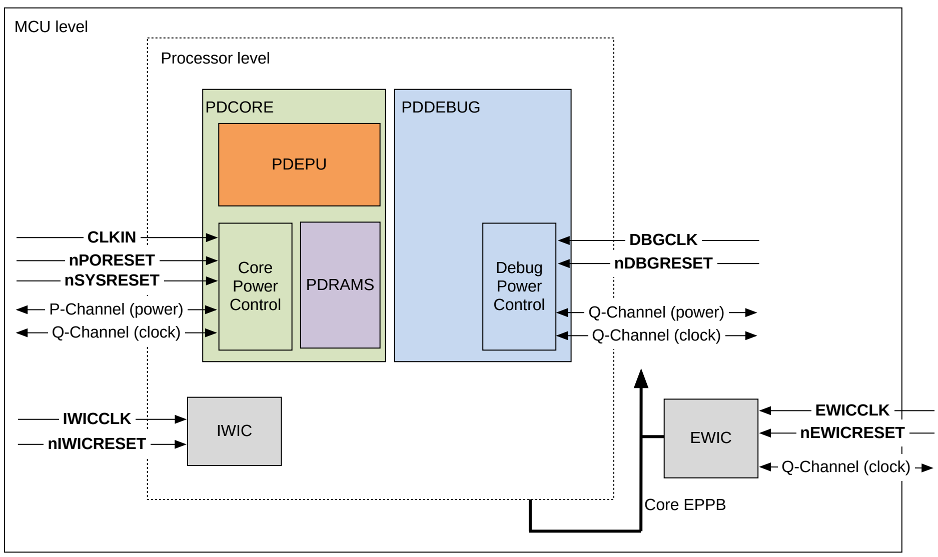
7 Power management

This chapter introduces Cortex®-M85 processor power management concepts.

7.1 Power domains

The Cortex®-M85 processor can be partitioned into power domains as shown in the following figure.

Figure 7-1: Cortex®-M85 processor power domains



The power domains are described in the following table.

Table 7-1: Power domain description

Power Domain	Description
PDCORE	This contains the processor core, L1 memory system, the *Cross Trigger Interface* (CTI), *Nested Vectored Interrupt Controller* (NVIC), and Core *External Private Peripheral Bus* (EPPB) interface.
PDEPU	This contains all *Extension Processing Unit* (EPU) logic, that is, the floating-point and *M-profile Vector Extension* (MVE) logic.
PDRAMS	This contains the L1 instruction cache and data cache RAMs.
PDDEBUG	This contains most of the debug logic. It includes the *BreakPoint Unit* (BPU), *Data Watchpoint and Trace* (DWT), *Instrumentation Trace Macrocell* (ITM), *Embedded Trace Macrocell* (ETM), and Debug *External Private Peripheral Bus* (EPPB) interface.

- The *Internal Wakeup Controller* (IWIC) is located in a separate power domain, the IWIC power domain, that might be on when the processor core is powered down, to allow the detection of wakeup events.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The MCU level in the processor deliverable includes an example *External Wakeup Controller* (EWIC). The EWIC can be placed in any point in the system that is considered to be Always-on relative to the processor domain.
- The IP deliverable that is shipped does not support any power domains at the MCU level, and the MCU level is considered to be relatively Always-on to the processor domain. You can use the delivered MCU and customize your system to include appropriate power domains depending on your implementation.

Note

For more information on the MCU level, see the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document that is available to licensees only.

7.2 Power states

The power domains in the Cortex®-M85 processor can be in ON, OFF, or RET power states. The RET power state requires the processor logic to be implemented with state retention.

The following table shows the supported power states.

Table 7-2: Supported power states

Power state	Clocks running	Processor logic powered	Register and RAM contents retained	Reset asserted
ON	Yes/No	Yes	Yes	No
RET	No	No	Yes	No
OFF	No	No	No	Yes

The following table shows the permitted Cortex®-M85 processor power states for the power domains.

Table 7-3: Permitted power states for Cortex®-M85 processor power domains

Power state	PDCORE	PDEPU	PDDEBUG	PDRAMS
ON	Permitted	Permitted	Permitted	Permitted
RET	Permitted	Permitted	Not permitted	Permitted[2]
OFF	Permitted	Permitted	Permitted	Permitted

Not all power state combinations are permitted. The combination of PDCORE, PDRAMS, and PDEPU power states is called the power mode. PDDEBUG is independent of the other power domains. It can either be ON or OFF, regardless of the processor power mode.

[2] Retention in the PDRAMS domain is only supported when the processor is in the MEM_RET (Cache), FULL_RET (Cache), or LOGIC_RET (Cache) power modes.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

When a power domain is in the ON power state, if the clock is not running, then the domain is considered to be in low-power state.

7.3 Power and operating mode transitions

The Cortex®-M85 processor power modes are based on the Arm standard modes and encodings. The power modes are extended with operating modes, which control whether the L1 instruction and data caches in the PDRAMS domain are enabled.

The Arm standard modes and encodings are defined in the *Arm® Power Control System Architecture* specification. The *Arm® Power Control System Architecture* specification is a confidential document that is only available to licensees.

An external power controller controls the processor power and operating mode through the P-Channel. An external clock controller controls the Q-Channel allowing system-level clock gating. The P-Channel and the clock control Q-Channel are connected to the *Core Power Control* (CPC) in the PDCORE domain. The CPC manages the internal clocking and reset of the PDCORE, PDRAMS, and PDEPU domains. It supports the clock and reset signals that are described in C.1 Clock and clock enable signals on page 341 and C.2 Reset signals on page 341, and system-level clock gating. The processor indicates the minimum required power mode according to its state and internal control registers using the **COREPACTIVE** signal. For more information on **COREPACTIVE**, see C.17 P-Channel and Q-Channel power control signals on page 361 and 7.5.1 COREPACTIVE signal encoding on page 125.

An external power controller controls the debug power mode through the debug domain power control Q-Channel. The debug domain power control Q-Channel is connected to the *Debug Power Control* (DPC) in the PDDEBUG domain. A clock control Q-Channel is also available to support high-level clock gating.

Only certain transitions between power and operating modes are allowed. Figure 7-2: Permitted power and operating modes and transitions on page 118 shows these permitted transitions. If an external power controller request is made to move between two modes which are not directly connected, then the request is denied (using **COREPDENY**).

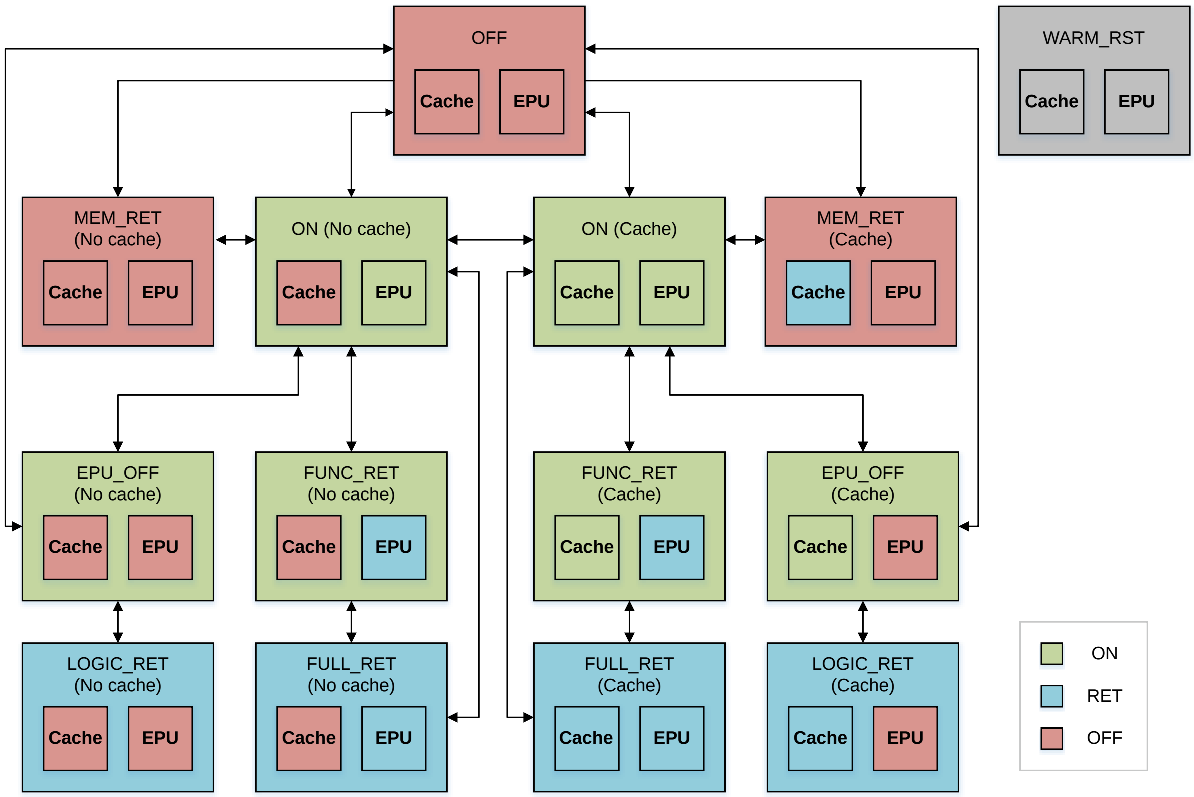
Retention in the PDRAMS domain depends on the overall power and operating mode. RAM retention is selected by entering any of the following:

- MEM_RET (Cache).
- FULL_RET (Cache).
- LOGIC_RET (Cache).

In other power modes, the PDRAMS state depends on the operating mode.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 7-2: Permitted power and operating modes and transitions



When the **COREPACTIVE** signal indicates a required move between two modes which are not directly connected, the external power controller must transition through one or more intermediate modes to get to the final required power and operating mode. When only a change in PDRAMS is required even if the change involves moving through multiple power and operating modes, the processor supports this and indicates the required intermediate transitions using the **COREPACTIVE** signals. See 7.3.1 Operating mode transitions which change PDRAMS power state on page 119.

The following table describes the power and operating modes that are shown in Figure 7-2: Permitted power and operating modes and transitions on page 118.

Table 7-4: Power and operating mode transitions

Power and operating mode	Description
ON (Cache)	Full Run mode with *Extension Processing Unit* (EPU) and cache powered on.
ON (No cache)	Full Run mode with EPU on and cache powered off.
FUNC_RET (Cache)	Run mode with EPU in software transparent low-power state but EPU state is retained and cache is powered on.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Power and operating mode	Description
FUNC_RET (No cache)	Run mode with EPU in software transparent low-power state but EPU state is retained. Cache is powered off (if present), or cache is not present.
EPU_OFF (Cache)	Run mode with EPU powered off. Save and restore of EPU state is required.
EPU_OFF (No cache)	Run mode with EPU and cache, if present, powered off. Save and restore of EPU state is required.
FULL_RET (Cache)	All functional logic and cache in retention. This is software transparent powerdown.
FULL_RET (No cache)	All functional logic in retention with cache powered off (if present), or the cache is not present. This is software transparent powerdown.
LOGIC_RET (Cache)	This is partially software-transparent powerdown. EPU has been powered off.
LOGIC_RET (No cache)	This is partially software-transparent powerdown. EPU has been powered off. Cache is powered off (if present), or cache is not present.
MEM_RET (Cache)	All functional logic is powered off, RAMs in retention.
MEM_RET (No cache)	MEM_RET (No cache) is functionally identical to OFF. The power mode and associated transitions are included for compatibility with the Arm CoreLink™ PCK-600 Power Control Kit PPU. The Cortex®-M85 processor never requests this state using the P-Channel **COREPACTIVE** output signal.
OFF	Powered off, Shutdown mode.
WARM_RST	Warm reset.

In Table 7-4: Power and operating mode transitions on page 118, the No cache operating mode implies that if your system configuration includes caches, then the cache is present, but disabled and powered OFF. The following register bits are set to appropriate values:

- MSCR.ICACTIVE and MSCR.DCACTIVE are 0.
- CPDLPSTATE.RLPSTATE is `0b11`.

Note

- A transition from OFF to MEM_RET is allowed. Arm recommends this as being required for full compatibility with the Arm CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU). Transitions from MEM_RET to OFF are not allowed. The system is responsible for maintaining power in the RAMs to ensure that processor cache content is preserved when entering MEM_RET.
- A transition from OFF to MEM_RET or MEM_RET to ON does not invalidate the cache even when **INITL1RSTDIS** is set to 0.
- MEM_RET (No cache) is functionally identical to OFF. The state and associated transitions are included for compatibility with current Arm CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU). The Cortex®-M85 processor never requests this state using the P-Channel **COREPACTIVE** output signal.
- A request on the P-Channel to transition to the current power mode is always accepted.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.3.1 Operating mode transitions which change PDRAMS power state

The processor supports transitions between operating modes where the PDRAMS domain is enabled or disabled.

For example, if the operating mode is ON (No cache) the processor can request to enable PDRAMS. This request results in **COREPACTIVE[16]** being asserted, requesting a transition to ON (Cache), but the other bits on **COREPACTIVE** remain static. The transition between ON (Cache) and ON (No cache) is called a change of operating mode.

The CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU) supports dynamic transitions between operating modes only when in the ON power mode. Therefore, when there is a request to change the operating mode (enable or disable the cache) for other active power modes like EPU_OFF and FUNC_RET, the processor drives **COREPACTIVE** to ON and the power controller transitions to ON. This allows the operating mode transition to occur. The *Core Power Control* (CPC) logic includes a secondary state-machine which transitions **COREPACTIVE** through the ON power mode to allow the external power controller to enable or disable PDRAMS.

For example, to transition from EPU_OFF (No cache) to EPU_OFF (Cache) the following steps need to take place. In this example, the processor starts in EPU_OFF(No Cache) mode, with **COREPACTIVE[6]** set HIGH indicating this is the current minimum required power mode. When the cache is enabled, the following steps need to be followed for the transition to take place:

1. The CPC drives **COREPACTIVE[8]** and **COREPACTIVE[16]** HIGH.
2. The external power controller responds with **COREPREQ** and **COREPSTATE** = ON (No cache). **COREPACTIVE[16]** is ignored because an operating mode transition cannot occur unless the power mode is ON.
3. The processor transitions the power mode from EPU_OFF (No cache) to ON (No cache).
4. The CPC continues to drive **COREPACTIVE[8]** and **COREPACTIVE[16]** HIGH.
5. The external power controller responds with **COREPREQ** and **COREPSTATE** = ON (Cache), requesting a change in operating mode in the ON power mode.
6. The processor transitions the power mode from ON (No cache) to ON (Cache).
7. Power mode stays at ON mode when automatic cache invalidation is in progress. Requested power modes will be denied until automatic cache invalidation completes in ON (cache).
8. The CPC deasserts **COREPACTIVE[8]**. **COREPACTIVE[16]** and **COREPACTIVE[6]** continue to be driven HIGH, indicating EPU_OFF (Cache) as the minimum required power mode.
9. The external power controller responds with **COREPREQ** and **COREPSTATE** = EPU_OFF (Cache).
10. The processor transitions the power mode from ON (Cache) to EPU_OFF (Cache) and continues to assert **COREPACTIVE[6]** and **COREPACTIVE[16]** HIGH.

Tip

For more information on the **COREPACTIVE** output signal encoding, see 7.5.1 COREPACTIVE signal encoding on page 125.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.4 Core P-Channel and power mode selection

The power modes are based on the power state of the PDCORE, PDEPU, and PDRAMS domains.

The requested power mode is defined according to the lowest achievable mode based on the processor logic state, external conditions, and the corresponding CPDLPSTATE register fields. The resulting power mode is driven on the P-Channel **COREPACTIVE** output signal.

Tip

For more information on **COREPACTIVE** signal encoding, see 7.5.1 COREPACTIVE signal encoding on page 125.

The following table shows the resultant overall power mode that is based on the requests from each individual processor power domain.

Table 7-5: Requested domain power states and resultant power and operating mode

Requested domain power states			Resultant power and operating mode
PDCORE	PDEPU	PDRAMS	
ON	ON	ON	ON (Cache)
ON	ON	OFF	ON (No cache)
ON	RET	ON	FUNC_RET (Cache)
ON	RET	OFF	FUNC_RET (No cache)
ON	OFF	ON	EPU_OFF (Cache)
ON	OFF	OFF	EPU_OFF (No cache)
RET	RET/ON	ON	FULL_RET (Cache)
RET	RET/ON	OFF	FULL_RET (No cache)
RET	OFF	ON	LOGIC_RET (Cache)
RET	OFF	OFF	LOGIC_RET (No cache)
OFF	ON	ON	ON (Cache)
OFF	ON	OFF	ON (No cache)
OFF	RET	ON	FULL_RET (Cache)
OFF	RET	OFF	FULL_RET (No cache)
OFF	OFF	ON	MEM_RET (Cache)
OFF	OFF	OFF	OFF
-	-	-	WARM_RST

Some combinations of power domain states do not map directly onto a power mode:

- Requesting ON for PDEPU when PDCORE is RET always results in a power mode with the EPU in retention.
- If PDEPU is required to be ON or RET, the selected power mode always retains EPU state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The lowest possible power mode is selected which matches the requested PDRAMS power state.

At Cold reset, the internal power mode is OFF and the P-Channel **COREPACTIVE** signal is also driven OFF. Before fetching the reset vector or starting to execute instructions, the processor waits for the system to request or initialize an operational state for the PDCORE domain.

The following power modes are supported on the P-Channel for device state initialization at reset deassertion:

- OFF.
- MEM_RET.
- EPU_OFF.
- ON.

A period t_{init} is defined in device clock cycles after which the device is guaranteed to have sampled the P-Channel **COREPSTATE** input signal for all possible valid reset states. For the Cortex®-M85 processor, t_{init} is three cycles of **CLKIN**.

7.4.1 P-Channel interface tie-off when P-Channel is not used

When the P-Channel is not used in the system, there are some tie-off requirements that must be met.

The following table shows the P-Channel interface tie-off when P-Channel interface is not used.

Table 7-6: P-Channel interface tie-off when P-Channel interface is not used

P-Channel signal	Tie-off values when P-Channel interface is not used
COREPSTATE	The value can be any of the following: • `0b11000`, indicating the power and operating mode is ON (Cache) • `0b10110`, indicating the power and operating mode is EPU_OFF (Cache) • `0b01000`, indicating the power and operating mode is ON (No cache) • `0b00110`, indicating the power and operating mode is EPU_OFF (No cache)
COREPREQ	`0b0`

If the P-Channel is not used in the system and the interface input signals are tied-off, the processor transitions to ON or EPU_OFF power mode out of Cold reset and starts executing instructions.

Note

COREPSTATE must be configured only to ON (No Cache) or EPU_OFF (No cache) if the instruction and data caches have not been configured in the processor.

The parameters `ICACHESZ[4:0]` and `DCACHESZ[4:0]` must be set to `0b00000`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Otherwise, processor behavior is **UNPREDICTABLE**.

For more information on these parameters, see *Arm® Cortex®-M85 Processor Integration and Implementation Manual*.

The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document available to licensees only.

7.5 COREPACTIVE and required power mode

The *Core Power Control* (CPC) unit in the PDCORE power domain determines the required minimum power mode and drives this mode on the P-Channel **COREPACTIVE** output signal.

Tip

For more information on the **COREPACTIVE** output signal encoding, see 7.5.1 COREPACTIVE signal encoding on page 125.

The required power mode is a combination of the processor state and the CPDLPSTATE register. This combination allows software to select the required low-power state for each power domain.

For more information on the CPDLPSTATE register, see 5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register on page 89.
The CPDLPSTATE register controls the three types of low-power state. The low-power states are:

- OFF
- RET
- ON with the clock off

Note

If present, external coprocessors are included in the requirements for moving the PDCORE domain to low-power state.

The CPDLPSTATE register can be used to select low power states based only on stopping the clock input to the PDCORE domain, **CLKIN**. The Q-Channel that is associated with **CLKIN** drives the **CLKINQACTIVE** signal LOW providing a hint to the system that the **CLKIN** Q-Channel might accept a quiescence request, therefore, allowing the clock to be gated if:

- All the low-power requirements for the PDCORE and PDEPU domains are true apart from the value of CPDLPSTATE.
- The CPDLPSTATE fields CLPSTATE and ELPSTATE are not `0b00`.

The individual required power states are translated to one of the overall power modes that are given in Table 7-5: Requested domain power states and resultant power and operating mode

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

on page 121 and used to drive the **COREPACTIVE** signal. The following table describes the **COREPACTIVE** and **COREPSTATE** bits encoding.

Table 7-7: COREPSTATE and COREPACTIVE bits encodings

Processor power mode	Standard power mode	COREPSTATE[4] (With cache)	COREPSTATE[3:0]	COREPACTIVE[16] (With cache)	COREPACTIVE[8:0] most significant set bit
WARM_RST	WARM_RST	-	`0b1001`	-	-
ON (Cache)	ON	1	`0b1000`	1	Bit 8 = 1
ON (No cache)	ON	0	`0b1000`	0	Bit 8 = 1
FUNC_RET (Cache)	FUNC_RET	1	`0b0111`	1	Bit 7 = 1
FUNC_RET (No cache)	FUNC_RET	0	`0b0111`	0	Bit 7 = 1
EPU_OFF (Cache)	MEM_OFF	1	`0b0110`	1	Bit 6 = 1
EPU_OFF (No cache)	MEM_OFF	0	`0b0110`	0	Bit 6 = 1
FULL_RET (Cache)	FULL_RET	1	`0b0101`	1	Bit 5 = 1
FULL_RET (No cache)	FULL_RET	0	`0b0101`	0	Bit 5 = 1
LOGIC_RET (Cache)	LOGIC_RET	1	`0b0100`	1	Bit 4 = 1
LOGIC_RET (No cache)	LOGIC_RET	0	`0b0100`	0	Bit 4 = 1
MEM_RET (Cache)	MEM_RET	1	`0b0010`	1	Bit 2 = 1
OFF	OFF	-	`0b0000`	**COREPACTIVE** is driven to 0.	

Note

- **COREPACTIVE[16]** indicates the minimum required cache state. If the cache operating mode is required, **COREPACTIVE[16]** is HIGH.
- **COREPACTIVE** bits 0, 1, 3, 9-15, and 17-20 are not used. They are always tied LOW.
- **COREPSTATE** values not listed in Table 7-7: COREPSTATE and COREPACTIVE bits encodings on page 124 are invalid. If a system attempts to transition to one of these encodings, the P-Channel responds with **COREPDENY**.
- For more information on WARM_RST, see 7.9 Warm reset power mode on page 127.
- Power modes WARM_RST and OFF are independent from **COREPSTATE[4].** The processor behaves identically whether this bit is 1 or 0.
- The processor uses a different name for the MEM_OFF encoding in the *Arm® Power Policy Unit Architecture Specification* because the corresponding power mode affects the EPU rather than memory, but maintains compatibility with the PPU power mode.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.5.1 COREPACTIVE signal encoding

The following table shows the **COREPACTIVE** signal encoding.

Table 7-8: COREPACTIVE signal encoding

Signal bit	Encoding
[20:17]	Unused
[16]	Indicates requirement for cache ON state
[15:9]	Unused
[8]	ON
[7]	FUNC_RET
[6]	EPU_OFF
[5]	FULL_RET
[4]	LOGIC_RET
[3]	Unused
[2]	MEM_RET
[1]	Unused
[0]	OFF **Note:** Indicates that no bits are set.

7.6 PDCORE low-power requirements

The following conditions must be true to request a PDCORE low-power state on the **COREPACTIVE** signal using the P-Channel:

- The processor is in sleep mode.
- SCR.SLEEPDEEP is set.
- **WICCONTROL[0]** is asserted so that **SLEEPDEEP** means *Wakeup Interrupt Controller* (WIC) sleep.
- If *External Wakeup Interrupt Controller* (EWIC) is configured, any automatic WIC loading must be completed.
- The *Slave AHB* (S-AHB) interface is inactive.
- There is no **DEBUG AHB** (D-AHB) access to the PDCORE memory space.
- No MBIST operation is in progress.
- The processor core is not halted.
- CPDLPSTATE.CLPSTATE is not equal to `0b00`.

When the PDCORE low-power requirements are met, CPDLPSTATE.CLPSTATE selects the low-power state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- The input signal **CPSPRESENT[n]** indicates that coprocessor n is included and CPACR_S.CPn and CPACR_NS.CPn indicate that coprocessor n is enabled and needs power. For more information on CPACR, see the *Arm®v8-M Architecture Reference Manual*.
- If a coprocessor CPn that is included in the system is indicating that the state cannot be lost (**CPSPRESENT[n]**&&CPPWR.SUn=`0b0`), then a request to powerdown in CPDLPSTATE.CLPSTATE is converted to RET to preserve the coprocessor state. For more information on CPPWR, see the *Arm®v8-M Architecture Reference Manual*.
- To request a PDCORE low-power state using clock gating only, CPDLPSTATE.CLPSTATE must be `0b01`.

7.7 PDEPU low-power requirements

The following conditions must be true to request a PDEPU low-power state on the **COREPACTIVE** signal using the P-Channel:

- The processor core is not halted.
- There are no scalar floating-point or *M-profile Vector Extension* (MVE) instructions in progress.
- CPDLPSTATE.ELPSTATE is not equal to `0b00`.

When the PDEPU low-power requirements are met, CPDLPSTATE.ELPSTATE selects the low-power state.

Depending on the system, the use of FUNC_RET as a transparent Power down mode might result in excessive switching of the EPU between retention state and on state driven by the execution of floating point or MVE instructions in the processor pipeline. The automatic power switching sequences can take a significant number of cycles and cause delays in execution reducing performance. This operating mode should only be used if the system provides appropriate hysteresis so that power mode transitions are minimised during regular operation.

An alternative approach is to set CPDLPSTATE.ELPSTATE to 0b11 (OFF). If EPU state must be retained, specified by CPPWR.SU10 set to 0b0, then when no MVE or floating-point instructions are executing the EPU will enter an appropriate low-power state determined by the low-power state required for PDCORE:

- If PDCORE enters a low-power state that is, sleep using WFI or WFE then the EPU enters retention
- If PDCORE does not enter a low-power state, the EPU remains ON but the internal clocks are gated to minimise dynamic power.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- If CPPWR.SU10 is `0b0` and PDCORE is in a low-power state, then selecting OFF in CPDLPSTATE.ELPSTATE results in RET state being selected to prevent the state from becoming **UNKNOWN**.
- If CPPWR.SU10 is `0b0` and PDCORE is not in a low-power state, then selecting OFF in CPDLPSTATE.ELPSTATE results in ON (Clock Off) state being selected to prevent the state from becoming **UNKNOWN**. For more information on CPPWR, see the *Arm®v8-M Architecture Reference Manual*.

7.8 PDRAMS powerdown requirements

The following conditions must be true to powerdown PDRAMS:

- MSCR.DCACTIVE is equal to `0b0`. This field is ignored for transparent retention of the RAMs.
- MSCR.ICACTIVE is equal to `0b0`. This field is ignored for transparent retention of the RAMs.
- CPDLPSTATE.RLPSTATE is equal to `0b11`.
- No cache maintenance operation is in progress.
- Automatic cache invalidation is not active.
- No MBIST operation is in progress to the instruction cache or data cache.

The low-power state is selected using CPDLPSTATE.RLPSTATE.

7.9 Warm reset power mode

The WARM_RST power mode is used when external control logic requires the processor to be put in a safe state for Warm reset.

Asserting Warm reset (**nSYSRESET**) is allowed when PDCORE is in power mode OFF or MEM_RET. Applying the reset in any other mode (except for WARM_RST mode) will have an **UNPREDICTABLE** effect.

Asserting **nSYSRESET** when PDCORE is in an active state and not in WARM_RST state might result in system deadlock.

Entering WARM_RST

WARM_RST can only be entered when the PDCORE and PDEPU domains are powered on, corresponding to the ON power mode. Requesting WARM_RST from any other power mode results in **COREPDENY** being asserted.

The processor asserts **COREPACCEPT** when PDCORE is transitioning to a quiescent state, and is held asserted until core quiescence is achieved. Therefore, it is only safe to assert **nSYSRESET** after the P-Channel transition to WARM_RST is completed.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

This core quiescence requires that there are no outstanding transactions on the *Master-AXI* (M-AXI), *Peripheral AHB* (P-AHB), *Core External Private Peripheral Bus* (Core EPPB), and *Slave AHB* (S-AHB) interfaces. If a request is made on the S-AHB interface while the processor is in WARM_RST power mode it is ignored. Therefore, the system is responsible for ensuring that no accesses are made on the S-AHB slave interface until the processor leaves WARM_RST whether or not reset is asserted in the power mode.

The processor ensures that all the outputs of the PDCORE domain are set to their reset values. Therefore, when **nSYSRESET** is asserted these values do not change, which helps to prevent reset domain crossing issues.

In particular, the AIRCR.SYSRESETREQ is cleared on entry to WARM_RST, so that the **SYSRESETREQ** output signal is driven to 0 matching the reset condition.

Warm reset can always be applied safely when the processor is in a low-power sleep state with all power domains powered-on and no requests are active on the S-AHB or D-AHB interfaces.

If your system has a P-Channel interface for power control, then it is only safe to assert **nSYSRESET** when the processor is in any of the following modes:

- WARM_RST, which is advantageous because it does not require software support
- OFF
- MEM_RET

If your system does not use a P-Channel interface for power control, then Arm recommends that you assert **nSYSRESET** when the processor core is in sleep mode, all the power domains are powered up, and there are no S-AHB or D-AHB requests.

The Warm reset request does not require that any of the power domains change power state. The combination of power states remains unchanged from when the processor entered the WARM_RST power mode.

Exiting WARM_RST

The processor can exit WARM_RST mode, whether or not **nSYSRESET** has been asserted to reset the PDCORE power domain. If no reset has occurred program execution continues from where it was before WARM_RST was requested.

The processor asserts **COREPACCEPT** for any request to transition from WARM_RST to the ON power mode. Requests to transition from WARM_RST to any other power mode results in **COREPDENY** being asserted.

The WARM_RST request does not require that any of the power domains change power state. The combination of power states when in the WARM_RST power mode will be the same as before it entered that power mode. The **COREPACTIVE** output signal will remain the same value as it was before **COREPACCEPT** was asserted for **COREPSTATE** indicating WARM_RST entry.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

The Cortex®-M85 processor has internal logic that deals with any metastability caused by either of the following asynchronous resets:

- Asserting **nSYSRESET** while the processor core is in the WARM_RST, OFF, or MEM_RET power modes.
- Resetting any power domain because of entry to a power state that is controlled by the P-Channel or Q-Channel.

7.10 Debug Q-Channel and PDDEBUG power domain

A Q-Channel interface controls the PDDEBUG power domain.

The PDDEBUG power domain logic drives the **PWRDBGQACTIVE** signal HIGH to indicate that the domain is active if any of the following conditions are met:

- Trace is enabled, DEMCR.TRCENA=1.
- If configured, the *Embedded Trace Macrocell* (ETM) is enabled, TRCPDCR.PU=1.
- There is outstanding trace data in the ETM, *Instrumentation Trace Macrocell* (ITM), or *Data Watchpoint and Trace* (DWT).
- There is an outstanding access to any core resource from the *Debug AHB* (D-AHB) interface in the PDDEBUG power domain.
- The *Debug AHB* (D-AHB) interface is active.
- There is an outstanding access to any of the registers in PDDEBUG from software.
- If the CTI is enabled, and there is a valid mapping that is set up for an external cross trigger to be passed onto the processor, or CTI integration mode is enabled in CTI_ITCONTROL.
- The *BreakPoint Unit* (BPU) is enabled, FP_CTRL.ENABLE=1.
- DPDLPSTATE.DLPSTATE is `0b00` or `0b01`.

Note

- Setting DPDLPSTATE.DLPSTATE to `0b01` indicates that **DBGCLK** can be gated when the domain is idle. This results in the **DBGCLKQACTIVE** signal being set LOW when the PDDEBUG domain is idle.
- For more information on the DPDLPSTATE register, see 5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register on page 90.
- For more information on TRCPDCR, see *Arm® CoreSight™ ETM-M85 Technical Reference Manual*.
- For more information on the FP_CTRL and DEMCR, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on the Q-Channel interface and its signals, see the *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.11 Q-Channel clock control

To optimize power usage, the Cortex®-M85 processor includes Q-Channel interfaces which allow the system to gate the clocks that are associated with the PDCORE and PDDEBUG power domains at a high level in the clock tree.

The PDCORE clock signal, **CLKIN**, is controlled using:

- **CLKINQREQn**.
- **CLKINQACCEPTn**.
- **CLKINQDENY**.
- **CLKINQACTIVE**.

The PDDEBUG clock signal, **DBGCLK**, is controlled using:

- **DBGCLKQREQn**.
- **DBGCLKQACCEPTn**.
- **DBGCLKQDENY**.
- **DBGCLKQACTIVE**.

The following rules apply for PDCORE and PDDEBUG clock signals:

- If both **CLKIN** and **DBGCLK** are running, they must be fully synchronous to each other.
- **CLKINQACTIVE** is asserted when PDCORE requires a clock.
- **DBGCLKQACTIVE** is asserted when PDDEBUG requires a clock.
- **CLKIN** can only be gated when its clock control Q-Channel is in the *Q_STOPPED* state or when the PDCORE P-Channel is in LOGIC_RET, FULL_RET, MEM_RET, or OFF.
- **DBGCLK** can only be gated when its clock control Q-Channel is in *Q_STOPPED* state or when the PDDEBUG power control Q-Channel is in *Q_STOPPED*.

7.12 PWRCOREWAKEPACTIVE

The PDDEBUG domain asserts the **PWRCOREWAKEPACTIVE** output signal when there is a debugger (DAP) request to any IPPB and EPPB address spaces in PDCORE

This signal must be routed to the external power controller and used to power up the PDCORE domain. The processor uses an internal signal to determine when the core domain is active and when it is safe to perform the access. The **PWRCOREWAKEPACTIVE** signal can be combined with the **COREPACTIVE** signal to indicate to the external power controller that the PDCORE domain must be activated.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.13 PWRDBGWAKEQACTIVE

The PDCORE domain asserts the **PWRDBGWAKEQACTIVE** output signal for the following cases.

- When there is a Core request to any IPPB address spaces in PDDEBUG. These include DPDLPSTATE register (`0xE001E304`), ITM registers (`0xE0000000` – `0xE0000FFF`), DWT registers (`0xE0010000` – `0xE0010FFF`), BPU registers (`0xE0020000` – `0xE0020FFF`), and PMU registers (`0xE0030000` – `0xE0030FFF`).
- When there is a Core request to EPPB address spaces in PDDEBUG. These include ETM registers (`0xE0041000` - `0xE0041FFF`), CTI registers (`0xE0042000` - `0xE0042FFF`), and Processor ROM Table registers (`0xE00FF000` - `0xE00FFFFF`).
- When there is a Core request to Debug EPPB address spaces. These include TPIU (`0xE0040000` - `0xE0040FFF`), ETB (`0xE0045000` - `0xE0045FFF`), MCU external Debug EPPB interface (`0xE0049000` - `0xE00FEFFF`) and MCU level CoreSight ROM table base address.

This signal must be routed to the external power controller and used to power up the PDDEBUG domain. The processor uses an internal signal to determine when the debug domain is active and when it is safe to perform the access. The **PWRDBGWAKEQACTIVE** signal can be combined with the **PWRDBGQACTIVE** signal to indicate to the external power controller that the PDDEBUG domain must be activated.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8 Memory model

This chapter describes the Cortex®-M85 processor memory model.

8.1 Memory map

The default memory map for the Cortex®-M85 processor covers the range `0x00000000-0xFFFFFFFF`.

Table 8-1: Default memory map

Address Range (inclusive)	Region	Interface
`0x00000000-0x1FFFFFFF`	Code	All accesses are performed on the *Instruction Tightly Coupled Memory* (ITCM) or *Master-AXI* (M-AXI) interface.
`0x20000000-0x3FFFFFFF`	SRAM	All accesses are performed on the *Data Tightly Coupled Memory* (DTCM) or M-AXI interface.
`0x40000000-0x5FFFFFFF`	Peripheral	• Data accesses are performed on *Peripheral AHB* (P-AHB) or M-AXI interface. • Instruction accesses are performed on M-AXI.
`0x60000000-0x9FFFFFFF`	External RAM	All accesses are performed on the M-AXI interface.
`0xA0000000-0xDFFFFFFF`	External device	All accesses are performed on the M-AXI interface.
`0xE0000000-0xE00FFFFF`	PPB	• Instruction fetches are not supported. • Reserved for system control and debug. • Data accesses are either performed internally or on *External Private Peripheral Bus* (EPPB).
`0xE0100000-0xFFFFFFFF`	Vendor_SYS	• Instruction fetches are not supported. • `0xE0100000-0xEFFFFFFF` is reserved. Vendor resources start at `0xF0000000`. • Data accesses are performed on P-AHB interface.

Security states for memory requests

The AMBA® interfaces on the Cortex®-M85 processor include support for indicating the security level of a memory request for the following interfaces:

Table 8-2: Security signals used in Cortex®-M85 memory interfaces

Interface	AMBA® standard	Security signals
M-AXI	AMBA® 5 AXI	**ARPROT[1]**, **AWPROT[1]**.
P-AHB	AMBA® 5 AHB	**HNONSECP**
EPPB	AMBA® 4 APB	**PPROT[1]**

The security attribute of a memory request depends on the Security state of the processor and the regions defined in the internal *Secure Attribution Unit* (SAU) or an external *Implementation Defined Attribution Unit* (IDAU). However, in some areas of the memory map, the security level of data accesses are determined only by the Security state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

See the *Arm®v8-M Architecture Reference Manual* for more information about the memory model.

The TCM interfaces do not include signals indicating the security level of a transaction. Instead, the processor includes an internal security gate to support programmable regions, which conform to the *Trusted Base System Architecture* (TBSA) for Arm®v8-M. This security gating mechanism is described in 9.8 TCM and P-AHB security access control on page 146.

Bit-banding

This feature is not supported on the Cortex®-M85 processor unless your system includes additional hardware to perform the appropriate mapping. If bit-banding support is required, Arm recommends that peripherals are memory mapped to alias their bits to byte, halfword, or words.

8.2 Memory types

Each address in the memory map has a memory type which is determined by the default memory map or the *Memory Protection Unit* (MPU).

The memory types are:

Normal memory

By default, half of the memory space is classified as Normal memory. Normal memory has many attributes, including Cacheability (Non-cacheable, Write-Through Cacheable, Write-Back Cacheable) and Shareability (Inner Shareability and outer Shareability), that impacts how data can be used in the system. Unaligned accesses to this memory type are allowed. However, under software control, the processor can fault on Unaligned accesses to Normal memory.

Device memory

Device memory is not *idempotent* and it is generally used by peripherals.

Architecturally, memory locations that are *idempotent* have the following properties:

- Read accesses can be repeated with no side-effects.
- Repeated read accesses return the last value that is written to the resource being read.
- Read accesses can fetch additional memory locations with no side-effects.
- Write accesses can be repeated with no side-effects, if the contents of the location that is accessed are unchanged between the repeated writes or as the result of an exception.
- Unaligned accesses can be supported.
- Accesses can be merged before accessing the target memory system.

For more information, see the *Arm®v8-M Architecture Reference Manual*.

There are restrictions on how Device memory can be ordered, merged, or speculated. These restrictions subdivide Device memory into the following subtypes.

Gathering, G and nG

Gathering, G, is the capability to gather and merge requests together into a single transaction. nG represents the non-Gathering attribute.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Reordering, R and nR

Reordering, R, is the capability to reorder transactions. nR represents the non-Reordering attribute.

Early Write Acknowledgment, E and nE

Early Write Acknowledgment, E, is the capability to accept early acknowledgment of transactions from the interconnect. nE represents the non-Early Write Acknowledgement attribute, indicating that buffering is not permitted. For the Cortex®-M85 processor, nE Device transactions are buffered inside the processor itself. This attribute is then passed to the external interface to ensure that the response is received appropriately.

The Cortex®-M85 processor treats the different types of Device memory identically. However, for MVE instructions, regardless of the Gathering attribute, multiple requests might be merged into one transaction. For Device memory:

- Data accesses are coherent for all system observers.
- All accesses must be aligned to the data type specified in the instruction. Unaligned accesses generate an Alignment fault.

Remapping

The default memory map defines the Peripheral, External device, *Private Peripheral Bus* (PPB), and Vendor_SYS regions as Device and the rest of the memory regions as Normal.

- Normal memory can be changed to Device.
- Device memory can be changed to Normal except for the following cases.
    - The PPB region is always Device-nGnRnE.
    - The Vendor_SYS region is Device-nGnRE and can be changed to Device-nGnRnE.
    - Mapping the Vendor_SYS region from Device to Normal results in **UNPREDICTABLE** behavior.

For more information on memory types and their attributes, see the *Arm®v8-M Architecture Reference Manual*.

8.3 Private Peripheral Bus

The *Private Peripheral Bus* (PPB) memory region provides access to internal and external processor resources.

The following table shows the regions in the memory map where attributes are determined only by the Security state of the processor and cannot be controlled using the SAU or IDAU.

These regions are all associated with either *System Control Space* (SCS) or debug and trace components.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

All regions or peripherals listed in the following table contain CoreSight ID registers which are listed in the processor ROM table when the processor is configured to include the region or peripheral.

Table 8-3: IPPB memory region accesses

Address Range (inclusive)	Region or peripheral
`0xE0000000-0xE0000FFF`	*Instrumentation Trace Macrocell* (ITM), if configured to be included
`0xE0001000-0xE0001FFF`	*Data Watchpoint and Trace* (DWT)
`0xE0002000-0xE0002FFF`	*BreakPoint Unit* (BPU)
`0xE0003000-0xE0003FFF`	*Performance Monitoring Unit* (PMU)
`0xE0004000-0xE0004FFF`	Reserved
`0xE0005000-0xE0005FFF`	*Reliability, Availability, and Serviceability* (RAS) registers
`0xE0006000-0xE000DFFF`	Reserved
`0xE000E000-0xE000EFFF`	SCS
`0xE000F000-0xE001DFFF`	Reserved
`0xE001E000-0xE001FFFF`	IMPLEMENTATION DEFINED registers **Note:** The Security state of the processor controls these registers.
`0xE0020000-0xE002DFFF`	Reserved
`0xE002E000-0xE002EFFF`	SCS Non-secure alias
`0xE003E000-0xE003FFFF`	IMPLEMENTATION DEFINED registers Non-secure alias **Note:** The Security state of the processor controls these registers.

Table 8-4: EPPB memory region accesses

Address Range (inclusive)	Region or peripheral
`0xE0040000-0xE0040FFF`	*Trace Port Interface Unit* (TPIU)
`0xE0041000-0xE0041FFF`	*Embedded Trace Macrocell* (ETM), if configured to be included
`0xE0042000-0xE0042FFF`	*Cross Trigger Interface* (CTI)
`0xE0043000-0xE0044FFF`	Reserved
`0xE0045000-0xE0045FFF`	*Embedded Trace Buffer* (ETB), if configured to be included
`0xE0046000-0xE0046FFF`	Reserved
`0xE0047000-0xE0047FFF`	*External Wakeup Interrupt Controller* (EWIC), if configured to be included
`0xE0048000-0xE0048FFF`	Reserved
`0xE0049000-0xE00FEFFF`	*External Private Peripheral Bus* (EPPB) APB interface **Note:** Peripherals in the EPPB region can apply security checks by using the **PPROT[1]** signal to determine if the access was made from Secure or Non-secure state and respond with **PSLVERR** HIGH if the access is not allowed.
MCU level CoreSight™ ROM table base address-(MCU level CoreSight™ ROM table base address+`0xFFF`)	System-level ROM table **Note:** The base address of the system-level ROM table is implementation-dependent.
`0xE00FF000-0xE00FFFFF`	Processor ROM table

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.4 Unaligned accesses

The Cortex®-M85 processor has different levels of support for loads and stores to unaligned addresses. Unaligned accesses are less efficient than using aligned memory locations, because the processor must perform a series of transactions to construct the necessary result.

Non-MVE accesses

For non-MVE accesses the following terminology applies:

Access size

The size of the data specified by an instruction.

Unaligned access

An access is unaligned if the access size is not aligned with address of the access.

Table 8-5: Unaligned non-MVE accesses

Behavior and performance	Non-MVE accesses
Cortex®-M85 processor faulting behavior	Unaligned non-MVE accesses fault in the following scenarios: • When the access is to the *External Private Peripheral Bus* (EPPB) region. • When the access is to a memory region marked as Device. • When the Unaligned trap is enabled (CCR.UNALIGN_TRP=1). For more information on the CCR register, see the *Arm®v8-M Architecture Reference Manual*. • When the access instruction is an `LDM` or `STM`.
Performance implications	Unaligned non-MVE accesses might be result in multiple smaller transfers. Therefore, there is a potential performance impact.

MVE accesses

For MVE accesses the following terminology applies:

Element size

The size of the data specified by an instruction.

Unaligned access

An access is unaligned if the element size is not aligned with the address of the access.

Table 8-6: Unaligned MVE accesses

Behavior and performance	MVE accesses
Cortex®-M85 processor faulting behavior	Unaligned MVE accesses always raise a UsageFault exception.
Performance implications	If an MVE transaction is not aligned to 32 bits but is still considered to be an aligned MVE transaction, then there is a performance impact because MVE instructions always transfer 128 bits of data as multiple 32 bits data transactions.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

VLDRB, VLDRH, VLDRW examples

To illustrate unalignment in MVE accesses, consider the following Vector Load Register instruction examples:

Table 8-7: VLDRB, VLDRH, VLDRW examples

Syntax	Alignment and faulting behavior	Performance implications?
`VLDRH.S16 Q0, [R1, #0]`	Aligned access	No
`VLDRB.S8 Q0, [R0, #1]`	Aligned access	Yes
`VLDRW.S32 Q0, [R2, #1]`	Unaligned access, UsageFault occurs	-

Note

In the preceding examples, the base register values for `R1`, `R0`, and `R2` are aligned to the data type.

8.5 Access privilege level for Device and Normal memory

The AMBA® 5 AXI, AMBA® 5 AHB, and AMBA® 4 APB protocols include signals that allow the privilege level of an access to be reported to the system.

The Cortex®-M85 processor supports these signals across the *Master AXI* (M-AXI), *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (Core EPPB and Debug EPPB) interfaces for Device memory. It also supports privilege reporting for Normal memory on P-AHB. However, accesses to Normal memory on M-AXI can be buffered and cached so memory read and write requests and instruction fetches from both privileged and unprivileged software can be merged. For these transactions, the AXI signals **ARPROT[0]** and **AWPROT[0]** are always 1 indicating a privileged access. Access permission to a region of memory can always be restricted to software running in privileged mode by using the *Memory Protection Unit* (MPU).

The *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) interfaces provide signals **ITCMPRIV**, **D0TCMPRIV**, **D1TCMPRIV**, **D2TCMPRIV**, and **D3TCMPRIV** to indicate the privilege of all memory accesses.

For more information on these signals, see the C.7 Instruction Tightly Coupled Memory interface signals on page 346 and C.8 Data Tightly Coupled Memory interface signals on page 347.

8.6 Memory ordering and barriers

Transactions that are performed on different interfaces can be reordered relative to one another, even if one or more of them is to Device memory.

In this context, the *Internal Private Peripheral Bus* (IPPB) region must be considered as a distinct interface. Therefore, *Private Peripheral Bus* (PPB) accesses can be reordered relative to Device accesses performed on the *Peripheral AHB* (P-AHB) or *Master AXI* (M-AXI).

This is consistent with the architectural memory ordering requirements as defined in the *Arm®v8-M Architecture Reference Manual* based on the assumption that the same peripheral is never mapped onto multiple interfaces.

If stricter ordering is required between two transactions to different interfaces, a `DMB` or `DSB` instruction must be inserted between them. For transactions to the same interface, two transactions to Device memory are always performed in program order.

TCMs are always implicitly Normal memory and any attempt to enforce stricter requirements by changing *Memory Protection Unit* (MPU) attributes are ignored.

The Arm®v8.1-M architecture includes the load-acquire and store-release instructions. These can be used to implement hardware-level support for the C++11 standard library atomic operations.

ISB instructions are required to guarantee the effect of instructions during context changes because the processor can prefetch several instructions before they are executed.

8.7 Execute Only Memory

The Cortex®-M85 processor supports system level use of *eXecute Only Memory* (XOM) on the *Master AXI* (M-AXI) and *Tightly Coupled Memory* (TCM) interfaces. The system integrator is responsible for adding relevant system design logic to support use of XOM.

In an XOM configuration, memory that is designated as execute-only cannot be read directly or indirectly by software running on the processor, or by the debugger. XOM operation requires that software is compiled so that literals are constructed through instruction fetches rather than explicit loads from memory. For example, using the `MOVT` and `MOVW` instructions.

XOM on the TCM interfaces is supported by the **xTCMMASTER** output signal which is set to `0b0000` for instruction fetches from software running on the processor. Any access to an XOM region which is not recognized as an instruction fetch can be aborted by asserting the **xTCMERR** signal. XOM regions protected in this way can never be accessed by *Slave AHB* (S-AHB) as a read on the slave interface will always result in a TCM access with **xTCMMASTER** set to `0b0011`.

XOM on the AXI interface requires that instruction fetches can be identified on the AXI interface. This can be done by checking the AXI read ID, **ARPROT[2**] which is only asserted for instruction fetch requests. The processor supports direct access to the cache RAM, therefore, access to the

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

L1 instruction cache must also be restricted. This can be achieved by asserting the external input signal **LOCKDCAIC**.

For more information on XOM, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9 Memory Authentication

This chapter describes the *Memory Authentication Unit* (MAU) responsible for controlling access to memory.

9.1 MAU features

The *Memory Authentication Unit* (MAU) receives requests from units that perform memory accesses, and the MAU returns responses to these units. These responses are a combination of all the responses from the *Memory Protection Unit* (MPU), *Security Attribution Unit* (SAU), *Implementation Defined Attribution Unit* (IDAU), and *TCM Gate Unit* (TGU). The MAU contains the following units or interfaces to units.

- MPU. For more information, see the 9.3 Memory Protection Unit on page 142.
- TGU. For more information, see the 9.7 TCM Gate Units on page 146.
- SAU. For more information, see the 9.2 Security Attribution Unit on page 140.
- Interface to the IDAU. For more information, see the 9.4 Implementation Defined Attribution Unit on page 144.
- Interface to the *Load Store Unit* (LSU) from the MAU. The LSU makes MAU lookup requests for loads, stores, and *Preload Data* (PLD), linefills, evictions, stacking, and unstacking.
- Interface to the TCMs from the TGU. The TCMs make TGU requests through the *Slave AHB* (S-AHB) interface for *Direct Memory Accesses* (DMAs), unstacking requests, instruction fetches, and loads and stores from the processor. For more information, see the 10.8 TCM interfaces on page 178.
- Interface to the *Instruction Fetch Unit* (IFU) from the MAU. The IFU makes lookup requests for instructions and vector fetches.

Note

When changing security attribution of an address by either reprogramming the SAU or changing the external IDAU mappings, cache maintenance is required.

9.2 Security Attribution Unit

The *Security Attribution Unit* (SAU) provides security attribution for the Cortex®-M85 processor.

SAU features

- The SAU is a programmable unit that determines the security of an address.
- The number of regions that are included in the SAU can be configured in the Cortex®-M85 implementation to be 0, 4, or 8.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The SAU is not used for *Slave AHB* (S-AHB) accesses.

Exemptions and faults

- The *System Control Space* (SCS) and all debug components are exempt from security checking.
- Accesses that violate the security settings cause a SecureFault. In this case, any potential MemManage Fault is masked and the access on the bus is blocked.
- SecureFaults do not prevent Speculative accesses to the caches or TCMs, however, an access that faults never updates processor state.

Enabling the SAU

The SAU_CTRL.ENABLE determines whether programming the SAU affects the security of an address. For the Cortex®-M85 processor, this value resets to 0.

9.2.1 SAU register summary

The *Security Attribution Unit* (SAU) has various registers that are associated with its function.

Each of these registers is 32 bits wide. The following table shows the SAU register summary. See the *Arm®v8-M Architecture Reference Manual* for more information about the register addresses, access types, and reset values. All the registers in the following table are not banked between Security states.

Table 9-1: SAU register summary

Address	Name	Type	Reset value	Description
`0xE000EDD0`	SAU_CTRL	RW	`0x00000000`	SAU Control Register
`0xE000EDD4`	SAU_TYPE	RO	`0x0000000X` **Note:** SAU_TYPE[3:0] depends on the number of SAU regions included. This value can be 0, 4, or 8.	SAU Type Register
`0xE000EDD8`	SAU_RNR	RW	`0x000000XX`	SAU Region Number Register
`0xE000EDDC`	SAU_RBAR	RW	`0xXXXXXXX0`	SAU Region Base Address Register
`0xE000EDE0`	SAU_RLAR	RW	UNKNOWN	SAU Region Limit Address Register
`0xE000EDE4`	SFSR	RW	`0x00000000`	Secure Fault Status Register
`0xE000EDE8`	SFAR	RW	UNKNOWN	Secure Fault Address Register

9.2.2 Security levels

The security level that the SAU returns is a combination of the region type that is defined in:

- The internal SAU

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The associated external *Implementation Defined Attribution Unit* (IDAU)

The final security level uses the higher security level indicated by the SAU or IDAU.

When the SAU_CTRL.ENABLE is zero, the default internal security levels is selected by the SAU_CTRL.ALLNS field. In the Cortex®-M85 processor, the SAU_CTRL register resets to zero, setting all memory (apart from some specific regions in the PPB space) to Secure, and preventing any override of the security level by the IDAU.

The following table shows examples of how the final security level is chosen.

Table 9-2: Final security level selection examples

IDAU	SAU	Final security
Secure	Secure, Non-secure, or Non-secure Callable	Secure
Secure, Non-secure, or Non-secure Callable	Secure	Secure
Non-secure Callable or Non-secure	Non-secure Callable	Non-secure Callable
Non-secure Callable	Non-secure Callable or Non-secure	Non-secure Callable
Non-secure	Non-secure	Non-secure

For more information on the IDAU, see 9.4 Implementation Defined Attribution Unit on page 144.

9.3 Memory Protection Unit

The Cortex®-M85 processor supports Arm *Protected Memory System Architecture* (PMSA). The *Memory Protection Unit* (MPU) is primarily used for memory region protection.

MPU features

The MPU features include:

- Memory region protection.
- Access permissions.
- Exporting memory attributes to the system.
- The MPU is not used for *Slave AHB* (S-AHB) accesses.
- You can use the MPU to:
    - Enforce privilege rules.
    - Separate processes.
    - Manage memory attributes.

Permission and access violations

MPU mismatches and permission violations invoke the MemManage Fault handler. These violations result in MemManage Faults and the access on the bus is blocked. For more information on MemManage Faults, see the *Arm®v8-M Architecture Reference Manual*. MemManage Faults do not

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

prevent Speculative accesses to the caches or TCMs, however, an access that faults never updates processor state.

MPU configuration

The MPU can be configured to support 0, 4, 8, 12, or 16 memory regions.

Memory protection can be duplicated between Secure and Non-secure MPU (MPU_S and MPU_NS).

The number of regions in the Secure and Non-secure MPU can be configured independently, and each can be programmed to protect memory for the associated Security state.

9.3.1 Memory Protection Unit register summary

The *Memory Protection Unit* (MPU) has various registers that are associated with its function.

See the *Arm®v8-M Architecture Reference Manual* for more information about the register addresses, access types, and reset values. All the registers in the following table are banked between Security states.

Table 9-3: MPU register summary

Address	Name	Type	Reset value	Description
0xE000ED90	MPU_TYPE	RO	0x0000xx00 **Note:** MPU_TYPE[15:8] depends on the number of MPU regions configured. This value can be 0, 4, 8, 12, or 16.	MPU Type Register
0xE000ED94	MPU_CTRL	RW	0x00000000	MPU Control Register
0xE000ED98	MPU_RNR	RW	0x000000XX	MPU Region Number Register
0xE000ED9C	MPU_RBAR	RW	UNKNOWN	MPU Region Base Address Register
0xE000EDA0	MPU_RLAR	RW	UNKNOWN, bit [0] resets to 0.	MPU Region Limit Address Register
0xE000EDA4	MPU_RBAR_A1	RW	UNKNOWN	MPU Region Base Address Register Alias 1
0xE000EDA8	MPU_RLAR_A1	RW	UNKNOWN	MPU Region Limit Address Register Alias 1
0xE000EDAC	MPU_RBAR_A2	RW	UNKNOWN	MPU Region Base Address Register Alias 2
0xE000EDB0	MPU_RLAR_A2	RW	UNKNOWN	MPU Region Limit Address Register Alias 2

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE000EDB4`	MPU_RBAR_A3	RW	**UNKNOWN**	MPU Region Base Address Register Alias 3
`0xE000EDB8`	MPU_RLAR_A3	RW	**UNKNOWN**	MPU Region Limit Address Register Alias 3
`0xE000EDC0`	MPU_MAIR0	RW	**UNKNOWN**	MPU Memory Attribute Indirection Register 0
`0xE000EDC4`	MPU_MAIR1	RW	**UNKNOWN**	MPU Memory Attribute Indirection Register 1

9.4 Implementation Defined Attribution Unit

The Cortex®-M85 processor supports an external *Implementation Defined Attribution Unit* (IDAU) to allow the system to determine the security level that is associated with any given address.

- The processor has five external interfaces for the IDAU with identical signals, properties, and requirements.
  - An interface for instruction fetches and exception vector read operations.
  - Four interfaces for all other data read and write operations from load and store instructions, register stacking on exception entry and exit, and debug memory accesses.
- The IDAU is not used for *Slave AHB* (S-AHB) accesses.

Security levels

The security level that the *Memory Authentication Unit* (MAU) returns is a combination of the region type defined in the internal SAU and the security type from the IDAU. For more information, see 9.2 Security Attribution Unit on page 140.

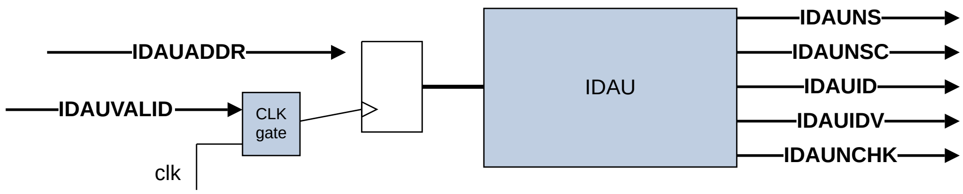
9.4.1 IDAU interface and backwards compatibility

Unlike previous Cortex®-M processors, the *Implementation Defined Attribution Unit* (IDAU) interface protocol in the Cortex®-M85 processor has a two-stage pipeline, allowing lookup, comparator, and resulting multiplexed logic to be balanced across a register slice to balance timing according to **IMPLEMENTATION-SPECIFIC** requirements.

The following figure shows how backwards compatibility can be implemented to allow for use with existing IDAU system designs.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 9-1: Cortex®-M85 IDAU interface backward compatibility



Note

To optimize your design, Arm recommends that the external IDAU is implemented with the processor logic to allow EDA tools to balance the timing of the IDAU logic with the internal *Security Attribution Unit* (SAU).

9.5 Memory regions not controlled by SAU and IDAU

The following address ranges in the memory map are regions whose attributes are determined only by the processor Security state and cannot be controlled using the *Security Attribution Unit* (SAU) or *Implementation Defined Attribution Unit* (IDAU).

Table 9-4: Memory regions not controlled by SAU and IDAU

Address range	Description
`0xE0000000-0xE00FFFFF`	This PPB memory region range is considered exempt for all SAU and IDAU accesses. This range is marked with the current Security state.
`0xF0000000-0xFFFFFFFF`	This range is always marked as Secure. It is not callable for Non-secure instruction fetches.

For more information about Security state, see the *Arm®v8-M Architecture Reference Manual*.

9.6 Security attribution signals

Security attribution is indicated for the Cortex®-M85 interfaces on the following signals:

- Bit [1] of **ARPROT** and **AWPROT** for the *Master AXI* (M-AXI) interface.
- **HNONSECP** for the *Peripheral AHB* (P-AHB) interface.
- **HNONSECD** for the *Debug AHB* (D-AHB) interface.
- **HNONSECS** for the *Slave AHB* (S-AHB) interface.

Using these signals ensures that the relevant interface components prevent Non-secure transfers to Secure memory or peripherals.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- S-AHB requests do not use the SAU and IDAU for security checking. However, **HNONSECS** is taken into consideration for security access gating using the *TCM Gate Unit* (TGU). See 9.8.2 Security access gating using the TGU on page 149.
- The security attribute depends on address of the location being accessed, and not on the Cortex®-M85 processor Security state that executes the load/store instructions or *Debug Access Port* (DAP) Security state that generates the debug request.
- Permitted DAP accesses to Secure *System Control Space* (SCS) registers in the range `0xE000E000-0xE000EFFF` are affected by the value of the following:
  - Secure debug enabled bit in the Debug Halting Control Status Register, DHCSR.S_SDE
  - Secure banked register select enable bit in Debug Security Control and Status Register, DSCSR.SBRSELEN
  - Secure banked register select bit in Debug Security Control and Status Register, DSCSR.SBRSEL
  - Current security state of the processor.

Note

Table 9-5: DAP accesses to Secure SCS registers

DHCSR.S_SDE	DSCSR.SBRSELEN	DSCSR.SBRSEL	Current Security state of the processor	View of the register accessed
0	-	-	-	Non-secure
1	0	-	Non-secure	Non-secure
1	0	-	Secure	Secure
1	1	0	-	Non-secure
1	1	1	-	Secure

For more information on DHCSR and DSCSR, see the *Arm®v8-M Architecture Reference Manual*.

9.7 TCM Gate Units

There are two *TCM Gate Units* (TGUs), one for *Instruction Tightly Coupled Memory* (ITCM) accesses (ITGU), and one for *Data Tightly Coupled Memory* (DTCM) accesses (DTGU), that are responsible for TCM security gating and control.

For more information on how the TGUs are responsible for security access control, see 9.8 TCM and P-AHB security access control on page 146.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9.8 TCM and P-AHB security access control

The Cortex®-M85 processor provides a mechanism to support further or a more fine-grained security access control on the TCM and *Peripheral AHB* (P-AHB) interfaces than provided by the SAU and IDAU.

This mechanism is compatible with the external gating mechanism described in *Arm® Platform Security Architecture Trusted Base System Architecture for Arm®v6-M, Arm®v7-M, and Arm®v8-M*.

To achieve additional security access control, you must use memory aliasing, configure the *Implementation Defined Attribution Unit* (IDAU) or *Security Attribution Unit* (SAU), and implement security gating.

Memory aliasing and IDAU and SAU configuration

Memory aliasing can be applied to the TCM and P-AHB interfaces. Memory aliasing is a duplication of all memory-mapped components in Secure and Non-secure address regions. These regions must be defined as Secure and Non-secure using the IDAU or SAU. For more information, see 9.8.1 Memory aliasing and IDAU/SAU configuration on page 147.

For more information on the SAU and IDAU, see 9.2 Security Attribution Unit on page 140 and 9.4 Implementation Defined Attribution Unit on page 144 respectively.

Security gating

The *TCM Gate Unit* (TGU) provides security gating for TCM accesses only. For more information on TGU security gating, see 9.8.2 Security access gating using the TGU on page 149.

To implement memory aliasing with the TCMs, you must use the TGU to maximize the benefits of the additional level of security that it provides.

If memory aliasing is not enabled (using the **CFGMEMALIAS** signal), the TGU is not used.

Accesses to the P-AHB require you to include your own external security gating logic.

Note

Additionally, memory aliasing can be done for the AXI interface, and all gating must be implemented externally. Therefore, the description of this behavior is outside the scope of this document. For more information, see the *Arm® Platform Security Architecture Trusted Base System Architecture for Arm®v6-M, Arm®v7-M, and Arm®v8-M*.

9.8.1 Memory aliasing and IDAU/SAU configuration

In normal operation, the TCM and *Peripheral AHB* (P-AHB) interfaces are mapped to regions in the memory map.

Code region
Base address `0x00000000` is used for *Instruction Tightly Couple Memory* (ITCM).

SRAM region
Base address `0x20000000` is used for *Data Tightly Couple Memory* (DTCM).

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Peripheral region
Base address `0x40000000` is used for P-AHB.

The TCM regions extend from their base to a limit that is defined by the physical size (in bytes) of the TCM set by the input signals **CFGITCMSZ** and **CFGDTCMSZ**. The P-AHB region extends from the base to its region size (in bytes) defined by the **CFGPAHBSZ** input signal.

Memory aliasing is enabled by tying the external input signal **CFGMEMALIAS[4:0]** to a non-zero value. The aliased address bit can be set from bit [24] to bit [28] using the **CFGMEMALIAS[4:0]** signal. The address bit that is used for memory alias is determined by the following options:

- `0b00001`, indicating that the alias bit is bit[24].
- `0b00010`, indicating that the alias bit is bit[25].
- `0b00100`, indicating that the alias bit is bit[26].
- `0b01000`, indicating that the alias bit is bit[27].
- `0b10000`, indicating that the alias bit is bit[28].

This results in:

- A second CODE and SRAM address region mapped to the ITCM and DTCM respectively.
- A second region in the Peripheral region to be mapped to the P-AHB interface.

`0b00000` indicates that there is no memory aliasing. Setting the address bit to any other value results in **UNPREDICTABLE** behavior.

For example, if you are using **CFGMEMALIAS[4:0]** for memory aliasing and you have set **CFGMEMALIAS[4:0]** to `0b10000` (bit [28] is used as the alias bit), **CFGPAHBSZ** should correspond to the actual size of the P-AHB region (in bytes):

- The base address of the P-AHB region is from `0x40000000-0x40000000+` size_in_bytes(**CFGPAHBSZ**)-1
- The alias address of the P-AHB region is from `0x50000000-0x50000000` + size_in_bytes(**CFGPAHBSZ**)-1.

The following table demonstrates an example of memory aliasing for the ITCM, DTCM, and P-AHB when the alias is configured for bit[28] of the address. The actual accessible TCM regions depend on the size of the TCM configured in the processor. In the following table, the size of the P-AHB region is limited to 256MB to avoid overlap with the alias at bit[28].

Table 9-6: Example TCM memory address aliasing

Address	Target region
`0x00000000-0x00FFFFFF`	ITCM
`0x10000000-0x10FFFFFF`	ITCM alias
`0x20000000-0x20FFFFFF`	DTCM
`0x30000000-0x30FFFFFF`	DTCM alias
`0x40000000-0x4FFFFFFF`	P-AHB
`0x50000000-0x5FFFFFFF`	P-AHB alias

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

Base and alias regions can overlap in the Peripheral region because the P-AHB interface can be mapped to the entire 512MB. However, Arm recommends that you avoid doing this because the behavior is **UNPREDICTABLE**. The aliasing logic only affects the target interface for P-AHB and TCM and it does not change the actual address. External security logic on this interface must mask the address accordingly to map the two aliased addresses to the same physical peripheral.

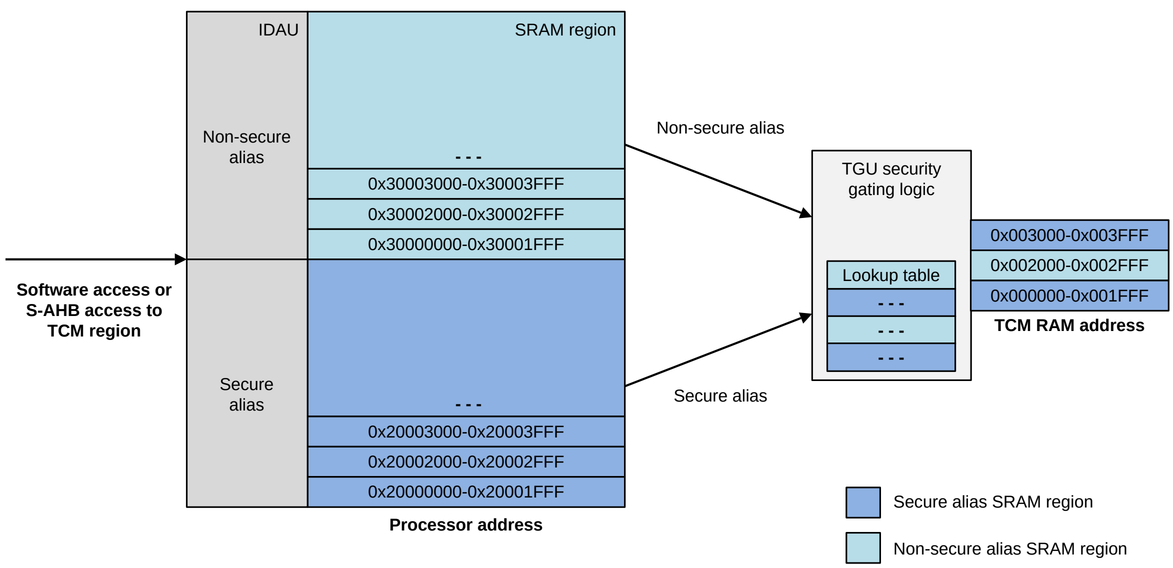
IDAU/SAU configuration for security access control

When memory aliasing is enabled, the *Implementation Defined Attribution Unit* (IDAU) or *Security Attribution Unit* (SAU) must be set up to map the two alias regions for each interface. This allows one region to be mapped as Secure and the other region to be mapped as Non-secure. This Secure and Non-secure mapping guarantees that software can access any given physical address in the TCM or P-AHB through external address mapping as either Secure or Non-secure regions.

For more information on setting up the IDAU using the relevant IDAU signals, see C.27 IDAU interface signals on page 367.

The following figure shows an example configuration of memory aliasing and IDAU configuration in the SRAM region and the DTCM using bit [28] of the address.

Figure 9-2: Example security alias and gating configuration on the DTCM



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9.8.2 Security access gating using the TGU

The *TCM Gate Unit* (TGU) is a security gate that allows the security attribute of a *Tightly Coupled Memory* (TCM) access to be checked against the security mapping for the address.

There are two optional TGUs, one for the *Instruction Tightly Coupled Memory* (ITCM) and one for the *Data Tightly Coupled Memory* (DTCM).

Each TCM is divided into blocks and a TGU lookup table is used to lookup the security mapping for an address. This is done in either of the following ways:

- For software accesses, the security mapping from the TGU lookup table is checked against the security attribute from the *Security Attribution Unit* (SAU) and *Implementation Defined Attribution Unit* (IDAU).
- For S-AHB accesses, the security mapping from the TGU lookup table is checked against the **HNONSECS** input signal which provides security level information for S-AHB accesses.

9.8.3 TGU configuration

Each *TCM Gate Unit* (TGU) is configured using the `xTGU`, `xTGUBLKSZ`, and `xTGUMAXBLKS` parameters.

Note

In this section, xTGU refers to *Instruction TCM Gate Unit* (ITGU) and *Data TCM Gate Unit* (DTGU).

The `xTGU` parameter configures the inclusion of the ITGU or DTGU, the `xTGUBLKSZ` parameter determines the block size, and `xTGUMAXBLKS` determines the maximum number of available blocks (which in turn defines the number of physical registers included in the TGU logic). The processor supports up to a maximum of 512 blocks for each TGU.

The `xTGUMAXBLKS` parameter is provided to allow a single processor implementation to support security gating across multiple different TCM size configurations using the external input signals **CFGITCMSZ** and **CFGDTCMSZ**.

You must configure `xTGUMAXBLKS` and `xTGUBLKSZ` to match the required range of TCM size. A TGU configuration is valid if both of the conditions in the following table are met.

Table 9-7: TGU configuration conditional validity

Condition	Formula
Block size * Maximum number of blocks = Maximum physical size of the TCM	$\texttt{xTGUBLKSZ} + \texttt{xTGUMAXBLKS} = \texttt{CFGxTCMSZ}_{\texttt{max}} + 4.$
Block size < Minimum physical size of the TCM	$\texttt{xTGUBLKSZ} < \texttt{CFGxTCMSZ}_{\texttt{min}} + 4$

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

This ensures that there are enough blocks to cover the largest TCM size and that at least two blocks cover the minimum TCM size. If these parameters are configured incorrectly, the TGU behavior becomes **UNPREDICTABLE**.

For a given processor implementation and integration, reading the xTGU_CFG.NUMBLKS and xTGU_CFG.BLKSZ register bitfields determines the number of available blocks in the lookup table and the block size respectively. For more information on these registers, see 5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers on page 100 .

When TCM gating is enabled, the Code and SRAM region of the processor memory map is aliased so that two regions map onto the same physical TCM address. These two regions should be mapped to different security levels. The security level attributed to the logical address used by software is always used to control the TGU. The two alias regions always map to the same physical address in the TCM memory.

The following table shows an example configuration where the processor ITGU is configured with 1KB blocks and supports a maximum ITCM size of 64KB and a minimum ITCM size of 4KB. In this case, ITGUMAXBLKS must be configured to `0b0110` or 64 blocks.

Table 9-8: Example TGU configuration for 1KB block size

ITCM size	CFGITCMSZ	ITGUBLKSZ	ITGUMAXBLKS	ITGU_CFG.NUMBLKS	ITGU_CFG.BLKSZ
4KB	0b0011	0b0101	0b0110	0b0010	0b0101
8KB	0b0100	0b0101	0b0110	0b0011	0b0101
16KB	0b0101	0b0101	0b0110	0b0100	0b0101
32KB	0b0110	0b0101	0b0110	0b0101	0b0101
64KB	0b0111	0b0101	0b0110	0b0110	0b0101

TGU block lookup table

Each block entry in the lookup table can be accessed by software using the xTGU_LUTn registers. Each register contains up to 32 block entries. For a valid block, the entry bit determines the required security level. All blocks reset to 0, therefore, at reset, all TCM memory is considered as Secure.

Any unused block entries in the lookup table, due to the configuration, do not affect the operation of the security gate and the corresponding xTGU_LUTn bitfield is RAZ/WI when accessed by software.

TGU enable and locking

TCM gating is enabled by tying the external input signal **CFGMEMALIAS** to a non-zero value.

The TGU can be locked from software access using the external input signals **LOCKITGU** and **LOCKDTGU**. When these signals are asserted the corresponding TGU registers become read-

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

only. This allows a TGU configuration to be programmed and then locked from further changes by software.

9.8.4 Security check and fault response

Accesses to a memory region that the TGU protects only proceed if the security level of the request matches the programmed security of the block. At reset, all blocks are Secure.

- Read requests on the external TCM interfaces are always Speculative, regardless of whether the access passes the security check in the TGU. Data from the RAM is always ignored if the check fails and the processor state is never updated.
- If the security check fails, write requests are always ignored and never carried out on the TCM interface.

The result of a security check mismatch in the TGU depends on the type of the access and the configuration of the ITGU_CTRL or DTGU_CTRL registers. The access is either ignored or generates a fault:

- A security check mismatch on an instruction fetch always results in a BusFault. The fault is recorded in AFSR.FTGU.
- If ITGU_CTRL.DBFEN or DTGU_CTRL.DBFEN is set, a security check mismatch on a data read or write results in a precise BusFault. The fault is recorded in AFSR.PTGU. If ITGU_CTRL.DBFEN or DTGU_CTRL.DBFEN is not set, no exception is raised.
- If ITGU_CTRL.DBGEN or DTGU_CTRL.DBEN is set, then a security check mismatch on a debug request causes **HRESP** to be asserted on the *Debug AHB* (D-AHB) interface. AFSR.PTGU is not updated on a security check mismatch from a debug request.
- If ITGU_CTRL.DEREN or DTGU_CTRL.DEREN is set, a security check mismatch on a read or write to the TCM from the S-AHB signals an error on the interface. For all mismatched read accesses, zero is returned to prevent any leaks of Secure data.

Note

If a data read access on the TCM returns an error on the interface (**ITCMERR** or **DTCMERR** input signal is asserted) for an address which fails the TGU security check and ITGU_CTRL.DBFEN or DTGU_CTRL.DBFEN is not set, then the overall behavior is RAZ/WI instead of raising a BusFault. This is consistent with a security fault response from the *Memory Authentication Unit* (MAU).

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10 Memory system

This chapter describes the Cortex®-M85 processor memory system.

10.1 Memory system features

The Cortex®-M85 processor memory system is an interface between the processor core and the cache RAMs, external memory interfaces and memory-mapped registers.

Note

For more information on how these units and interfaces interact with each other, see the Figure 3-1: Cortex-M85 processor block diagram on page 29.

Load Store Unit

The *Load Store Unit* (LSU) receives load and store accesses from the *Data Processing Unit* (DPU) and distributes these requests to the correct unit and returns any data or responses to the DPU.

Peripheral Interface Unit

The *Peripheral Interface Unit* (PIU) is responsible for the handling of stores to peripheral units *External Private Peripheral Bus* (Core EPPB and Debug EPPB), *Internal Private Peripheral Bus* (IPPB), and P-AHB. The PIU coordinates the following accesses to the peripheral busses.

- Loads from the LSU
- Stores from the STB

TCM Control Unit

The *TCM Control Unit* (TCU) arbitrates requests between the LSU and *Instruction Fetch Unit* (IFU), accesses the TCMs, and returns any data or responses to the requesting unit. The TCU contains a write queue for *Slave-AHB* (S-AHB) writes and a read prefetcher to improve the performance of 32-bit and 64-bit incrementing reads.

Tightly Coupled Memories

The Cortex®-M85 processor has two TCM memory types, the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM). There is one ITCM interface and four DTCM interfaces (D0TCM, D1TCM, D2TCM, and D3TCM respectively).

All the DTCM interfaces are 39 bits wide with 32 bits for data and 7 bits for *Error Correcting Code* (ECC). The ITCM interface is 72 bits wide with 64 bits for data and 8 bits for ECC.

ECC generation and correction logic can optionally be included for each TCM interface and an ECC error indication interface.

Memory accesses to the TCM, required for fetching instructions and for data transfer instructions, are performed if the address is in an enabled TCM region. Accesses that are not serviced by the TCM region are passed through the *Master-AXI* (M-AXI) interface or one of the peripheral interfaces.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Data Cache Unit

The *Data Cache Unit* (DCU) contains a four-way set-associative data cache and handles all accesses to this cache. These accesses include loads, stores, cache maintenance operations, evictions, and ECC error detection and correction.

The DCU can be configured to include logic to detect and process ECC errors.

Instruction Cache Unit

The *Instruction Cache Unit* (ICU) contains a two-way set-associative instruction cache, and it accepts instruction fetch requests from the IFU and returns data from either the instruction cache, the linefill buffer, or the BIU.

The ICU can be configured to include logic to detect and process ECC errors.

Store Buffer

The *STore Buffer* (STB) has six 64-bit slots that buffer stores to the AXI bus.

- The STB has five additional slots that are shared between the TCM and PIU writes.
- For Cacheable stores, the STB sends a lookup request to the DCU to see if the target address is in the cache. If it is, then the data is written directly to the cache once the *Bus Interface Unit* BIU allocates the tag and data chunk into the cache. If the target address is not in the cache and the access has a Write-Allocate hint, then the DCU makes a linefill request to the BIU. If the target address is not in the cache and it does not have a Write-Allocate hint, then the store is written out to the AXI bus.
- For Non-cacheable data, the data is written to the BIU write buffer.
- Write-Through stores are written out to the AXI bus even if they have been written into the cache.

Bus Interface Unit

The BIU contains one 32-byte write buffer and four Linefill Descriptors that can support four demand linefills. Six 64-bit AXI data return buffers are shared between the Linefill Descriptors to manage returning linefill data.

The BIU coordinates the following accesses to the M-AXI interface.

- Loads from the LSU
- Stores from the STB
- Evictions from the DCU
- Fetches from the IFU
- Linefills triggered by PLD instructions
- Speculative linefills triggered by the data prefetcher

Non-cacheable loads go directly to the AXI bus. Stores are buffered internally with the intention of being combined in a burst on the AXI. Cacheable Read-Allocate loads and Cacheable Write-Allocate stores trigger linefills and the data from the AXI bus is buffered in the linefill buffer until the line is complete and it can be allocated in the DCU.

The linefill buffers also buffer load data from Non-cacheable bursts.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

MBIST Interface Unit

The *MBIST Interface Unit* (MIU) provides the *Memory Built-In Self Test* (MBIST) interface.

The MBIST interface supports production MBIST.

M-AXI interface

The M-AXI interface is 64 bits wide and connects to the external memory system.

Peripheral-AHB interface

The PIU includes a 32-bit *Peripheral-AHB* (P-AHB) interface for accessing external peripherals.

Slave-AHB interface

The S-AHB interface is 64 bits wide and allows system accesses in and out of the TCMs.

PPB interfaces

The PIU includes the *Internal Private Peripheral Bus* (IPPB) interface to access internal PPB registers, and the *External PPB* (EPPB) APB interfaces to access external PPB registers.

10.2 Memory system faults

Memory system faults can occur on instruction fetches and data accesses.

Faults can occur on instruction fetches for the following reasons:

- *Memory Protection Unit* (MPU) MemManage fault.
- *Security Attribution Unit* (SAU) or *Implementation Defined Attribution Unit* (IDAU) SecureFault.
- BusFaults that are caused by an external AXI slave error (SLVERR), an external AXI decode error (DECERR), or corrupted transactions (RPOISON).
- TCM external error.
- Uncorrectable *Error Correcting Code* (ECC) errors in the TCM.
- Breakpoints and vector catch events.
- *TCM Gate Unit* (TGU) faults.

Faults can occur on data accesses for the following reasons:

- MPU MemManage fault.
- Alignment UsageFault.
- SAU or IDAU SecureFault.
- BusFaults that are caused by an external AXI slave error (SLVERR), an external AXI decode error (DECERR), or corrupted read data (RPOISON).
- BusFaults because of errors on the *External Private Peripheral Bus* (EPPB) APB interface.
- External AHB error from the *Peripheral-AHB* (P-AHB) interface.
- TCM external error.
- Uncorrectable ECC errors in the TCM or L1 data cache.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Watchpoints.
- *M-profile Vector Extension* (MVE) transactions, stacking, or unstacking to the *Private Peripheral Bus* (PPB) space.
- TGU faults.
- Unprivileged accesses to system registers which only privileged code can access.

10.2.1 Classes of fault

Faults can be classified as MemManage Faults, BusFaults, SecureFaults, and UsageFaults.

10.2.1.1 MemManage faults

The *Memory Protection Unit* (MPU) can generate a fault for various reasons.

For more information on MemManage Faults, see Permission and access violations on page 142 .

10.2.1.2 Bus faults

A memory access or instruction fetch performed through the *Master-AXI* (M-AXI) interface can generate different types of responses:

- Slave error (SLVERR).
- Decode error (DECERR).

AXI bus errors cause precise or imprecise BusFaults. Additionally, if the AMBA® 5 AXI signal, **RPOISON**, is asserted, an AXI read can generate a BusFault.

A memory access performed through the *Peripheral AHB* (P-AHB) interface can generate a single error response. The processor manages this in the same way as a response of SLVERR from the AXI interface.

Whether a memory or instruction fetch access on the TCM interface can be performed or not relies on the *TCM Gate Unit* (TGU), if implemented. Depending on the programming of the TGU, TGU faults can generate errors.

- For loads or stores, errors cause synchronous BusFaults.
- For read and write accesses from the *Slave AHB* (S-AHB) interface, an error causes an AHB slave error response on **HRESPS**. For writes, only TCM interface errors on ITCMERR or DTCMERR result in an imprecise error response on S-AHB through **SAHBWABORT**.

Synchronous BusFaults are generated in the following cases

- Instruction fetches.
- Data loads.
- Stores that generate a TGU fault.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Stores to PPB that cause a privilege violation.
- *M-profile Vector Extension* (MVE) stores and stacking to the PPB space.
- Uncorrectable *Error Correcting Code* (ECC) errors.

Asynchronous BusFaults are generated in the following cases:

- All stores except those that generate synchronous BusFaults.
- Dirty linefills that cause an AXI bus error.
- Unprivileged access to registers that can be accessed by privileged code only.

10.2.1.3 SecureFaults

If accesses do not pass the security attribution checks that the *Memory Authentication Unit* (MAU) performs, then a SecureFault is raised.

For more information on security attribution, see 3.1.3.1 Memory Authentication Unit on page 33.

Note

In most of the memory regions, debugger accesses are subject to validation and attribution. That is, the final Security state of an access on the *Master AXI* (M-AXI), indicated on **ARPROT[1]** and **AWPROT[1]** signals, the *Peripheral AHB* (P-AHB) interface, indicated on **HNONSECP** signal, or the *External Private Peripheral Bus* (EPPB) APB interfaces, indicated on **PPROT[1]** signal, is set by the *Security Attribution Unit* (SAU) in the same way as software generated accesses. The SAU blocks memory accesses which do not have the required permissions. For example, accesses to memory marked as Secure in the SAU when DHCSR.S_SDE is 0 or **HNONSECD** is HIGH. This results in an error response on the *Debug AHB* (D-AHB) interface, but unlike accesses that originate from software, a SecureFault is not raised.

10.2.1.4 Usage faults

UsageFault exceptions occur in the following cases:

- Any unaligned access when CCR.UNALIGN_TRP is set results in an UNALIGNED UsageFault exception. For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.
- Unaligned accesses to Device memory regions are not supported and result in an UNALIGNED UsageFault exception.
- Unaligned accesses from an instruction that does not support unaligned accesses result in an UNALIGNED UsageFault exception. For more information on these instructions, see the *Alignment behavior* section in the *Arm®v8-M Architecture Reference Manual*
- For *M-profile Vector Extension* (MVE) operations, a load or store access is considered unaligned if the address is not aligned to the specified element size. Using an address for an MVE load or store which is not aligned to the element size results in an UNALIGNED UsageFault being

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

raised. For more information on MVE and elements, see the *Arm®v8-M Architecture Reference Manual*.

- Accessing a coprocessor that does not exist results in a NOCP UsageFault.

Note

For more information on external coprocessors, see 13 External coprocessors on page 221. Additionally, for more usage restriction information, see 13.4 Coprocessor instruction restrictions on page 222.

10.3 Memory system behavior

The behavior of the memory system depends on the type attribute of the memory that is being accessed. Only Normal, cacheable memory regions can be cached in the RAMs.

The following points and the table that follows summarize the memory types and their associated memory system behavior:

- The memory system supports all memory types specified in the *Arm®v8-M Architecture Reference Manual*.
- For the data cache, all Shareable transactions are forced to be Non-cacheable because the Cortex®-M85 processor must be data coherent with other observers in the Shareability domain. On the data side, if a transaction is marked as Non-shareable, then caching can occur if the data cache is enabled (CCR.DC=1) and active (MSCR.DCACTIVE=1). For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*. For more information on MSCR, see 5.13 MSCR, Memory System Control Register on page 84.
- For the instruction cache, transactions marked as Shareable Cacheable are not forced to be Non-cacheable because the instruction cache cannot be dirty and its contents are always consistent with the external memory. Unless, the external memory changes, in which case, the instruction cache is invalidated. Therefore, caching occurs irrespective of the Shareability attribute. On the instruction side, caching can occur if, the instruction cache is enabled (CCR.IC=1) and active (MSCR.ICACTIVE=1). For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*. For more information on MSCR, see 5.13 MSCR, Memory System Control Register on page 84. The processor caches Shareable Cacheable instruction fetches, therefore, instruction cache software maintenance is always required for self-modifying code because only data access coherency is supported.
- The store buffer supports all stores to *Master-AXI* (M-AXI). It also handles the special behavior required for no Write-Allocate mode.
- All Shareable exclusive transactions to the M-AXI and *Peripheral AHB* (P-AHB) interfaces are marked as exclusive.
- All Non-shareable exclusive transactions to the M-AXI and P-AHB interfaces are not marked as exclusive.
- Only Normal memory is considered idempotent. For more information on the properties of idempotent Normal memory, see the *Normal memory* section *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- For exclusive accesses to Non-shared memory only the internal exclusive monitor is updated and checked. Exclusive accesses to Shared memory are checked using the internal and external monitor that uses the external memory interface M-AXI or P-AHB.

The following table summarizes the processor memory types and associated behavior for data accesses.

Table 10-1: Memory types and associated behavior for data accesses

Memory type	Device memory attributes	Shareability	Cacheability	Restartable	Exclusives handled
Normal	-	Shared	No Cacheability	Yes	Internal and external
	-	Non-shared	Only if memory attributes are Cacheable and the cache is present, enabled, and active[3].	Yes	Internal only
Device	Gathering, G and non-Gathering, nG	Yes	No	No	Internal and external
	Reordering, R and Non-Reordering, nR	Yes	No	No	
	Early Write Acknowledgment, E and No Early Write Acknowledgment, nE	Yes	No	No	

Note

- The Cortex®-M85 processor can merge accesses to Normal memory, but not to Device memory.
- An external interconnect can merge accesses to Normal memory, but must not merge accesses to Device memory.
- *M-profile Vector Extension* (MVE) instructions to Device memory might merge multiple accesses from the same micro-operation into one transaction, regardless of whether that memory has the Gathering attribute or not.

10.3.1 Speculative accesses

The Cortex®-M85 processor performs Speculative accesses to increase performance. The Arm®v8-M and Arm®v8.1-M architecture permit Speculative accesses. System designers must not assume that the scope of the speculation is fixed or definitively specified.

The following list describes some of the examples where Speculative accesses can occur:

- Speculative instruction fetches can be initiated to any Normal, executable memory address. This can occur regardless of whether the fetched instruction gets executed or, in rare cases, whether the memory address contains any valid program instruction.
- Speculative data reads can be initiated to any Normal, read/write, or read-only memory address. In some rare cases, this can occur regardless of whether there is any instruction that causes the data read.

[3] For more information on cache activity, see 10.9.6 Accessing the caches on page 189

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Speculative cache linefills can be initiated to any Cacheable memory address regardless of whether there is any instruction that causes the cache linefill.
- Speculative reads that target a TCM region can be initiated on any of the five TCM interfaces, regardless of which TCM interface the memory region is mapped to, or whether that address is mapped to any TCM interface.

However, Speculative accesses do not occur in the following cases:

- Speculative instruction fetches on the *Master AXI* (M-AXI) interface are never made to memory addresses in an Execute Never region.
- Speculative data cache linefills on the *Master AXI* (M-AXI) interface are never made to Non-cacheable memory addresses.
- Speculative data reads and Speculative cache linefills are never made to Device memory addresses.
- Speculative reads are never made on the *Peripheral AHB* (P-AHB) and *External Private Peripheral Bus* (EPPB) interfaces.
- Speculative writes are never made.

Note

Memory regions that are mapped to the TCM are always treated as Normal Memory and therefore are always subject to speculation.

10.3.1.1 Considerations for system design

The system designer must ensure that the system is robust enough to handle Speculative accesses, and all executable and Normal type memory regions are safe to access.

Preventing Speculative accesses

Speculative accesses do not cause any processor faults. The processor is aware whether an access is Speculative, and ignores any error response that the system signals because of the Speculative access. However, the system in which the processor is integrated in cannot distinguish between Speculative accesses and Non-speculative accesses. Therefore, the system designer is required to ensure that the system is robust enough to handle Speculative accesses, regardless of whether they are initiated to unexpected memory addresses.

Alternatively, if there are memory regions that are not mapped to the TCMs and to which Speculative access should not be initiated, Arm recommends setting those regions to have the following attributes with the *Memory Protection Unit* (MPU):

- Device
- Execute-never

In the Cortex®-M85 processor, the following conditions apply for speculative accesses:

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Speculation is not allowed for any access on M-AXI for Secure attributed memory regions without Secure access rights.
- Instruction fetches can be made speculatively on M-AXI to Normal memory that is not marked as execute-never.
- Data accesses to Normal memory can be speculative on M-AXI. In this case, speculative covers:
  - Linefills from the data prefetcher
  - Data accessed as part of the cache line beyond the specific locations accessed by the instruction
  - Linefills performed by non-faulting instructions speculatively executed but not committed
- Data accesses cannot be speculative on P-AHB or EPPB.
- Instruction fetches are not supported on P-AHB or EPPB.
- No external bus (M-AXI, P-AHB, EPPB) access are made for accesses encountering MPU, SAU or IDAU faults.

The TCMs are always treated as Normal memory. Therefore, they are always subject to speculation.

MPU, SAU, or IDAU violation behavior

On the M-AXI, P-AHB, or EPPB interfaces, an MPU, SAU, or IDAU violation is guaranteed to cause a fault and the access is not initiated on the interface. On the TCM interface, an MPU, SAU, or IDAU violation is guaranteed to cause a fault. However, a read access is still initiated, and in this case, the processor ignores the read data that is returned from the TCM.

10.3.2 Access privilege level for Device and Normal memory

The AMBA® 5 AXI, AMBA® 5 AHB, and AMBA® 4 APB protocols all include signals which allow the privilege level of an access to be reported to the system. The Cortex®-M85 processor supports these signals across the *Master AXI* (M-AXI), *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) interfaces for Device memory.

The Cortex®-M85 processor also supports privilege reporting for Normal memory on P-AHB. However, M-AXI accesses to Normal memory can be buffered and cached so memory read and write requests and instruction fetches from both privileged and unprivileged software can be merged. All M-AXI accesses to Normal memory are marked as privileged. For all M-AXI transactions, the AXI signals **ARPROT[0]** and **AWPROT[0]** are always 1 indicating a privileged access. Access permission to a region of memory can always be restricted to software running in privileged mode by using the *Memory Protection Unit* (MPU).

The following table shows the processor mode and privilege level values of the read channel protection signal. The security attributes of the transaction are stored in bit 1 of the **ARPROT** and **AWPROT** signal.

Table 10-2: Cortex®-M85 processor mode and read and write channel protection signal privilege information

Processor mode	Memory type	Value
-	Normal Cacheable	Always marked as Privileged

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Processor mode	Memory type	Value
-	Normal Non-cacheable	
Unprivileged	Device	Unprivileged
Privileged		Privileged

The instruction and data TCM interfaces provide signals **ITCMPRIV** and **D*TCMPRIV** to indicate the privilege of all memory accesses.

For more information on how security attributes are generated and determined, see 3.1.3.1 Memory Authentication Unit on page 33 .

10.4 Master-AXI interface

The *Master-AXI* (M-AXI) interface is a single 64-bit AMBA® 5 AXI interface for on-chip or off-chip memory and devices. The interface serves the memory regions that the TCM, *Peripheral AHB* (P-AHB), *Internal Private Peripheral Bus* (IPPB), and *External Private Peripheral Bus* (EPPB) interfaces do not cover.

The M-AXI interface can have either of the following configurations:

- High performance configuration.
- Area optimized configuration.

Both M-AXI configurations provide a store-buffer that supports data merging, reordering, and forwarding for Normal memory to minimize the number of AXI write transactions that are sent out to the system.

Note

- Implementing the L1 data cache results in the high-performance M-AXI configuration. When the L1 data cache is not present, the M-AXI defaults to the area optimized configuration.
- For more information on restrictions, see 10.4.6.1 Restrictions on AXI transfers on page 169.

10.4.1 High performance M-AXI configuration

The high performance M-AXI configuration supports extensive buffering and multiple outstanding AXI transactions to optimize memory system performance, even in the presence of large latencies.

This configuration includes a 4-way set associative L1 data cache that supports:

- Write-allocation.
- Write-Back.
- Write-Through.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Transient.

The cache supports automatic data prefetching that can be used for compute tasks that require large data sets that the TCMs cannot accommodate.

10.4.1.1 High performance configuration M-AXI attributes and transactions

The high performance configuration is designed to be used with a native AXI system with high memory bandwidth and support for multiple outstanding transactions. The following table shows the AXI attributes and transactions that the high performance M-AXI configuration supports.

Table 10-3: High performance configuration M-AXI attributes and transactions

AXI attribute	Value	Details
Write issuing capability	39	• 15 writes to Device memory • 24 writes to Normal memory, that can be evictions, write bursts, or single writes
Read issuing capability	9	• 4 data linefills, including linefills that the data prefetcher requests • 4 Non-cacheable data read • 1 instruction fetch or instruction linefill
Write ID capability	4	• 1 reserved for Device memory • 1 reserved for Normal Non-cacheable writes and exclusive writes • 1 reserved for Normal cacheable writes • 1 reserved for cache line evictions
Read ID capability	8	• 1 reserved for Normal Non-cacheable and Device memory • 6 reserved for data cache linefills • 1 reserved for instruction fetch or instruction linefill

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

AXI attribute	Value	Details
Combined issuing capability	48	• 39 outstanding writes • 9 reads from data linefills, Non-cacheable reads, and instructions fetches

Only a subset of all possible AXI transactions can be generated. These are:

- For Normal, Cacheable memory:
  - WRAP4 64-bit reads, for load, data prefetch and store linefills, and instruction linefills.
  - INCR4 64-bit writes, for evictions.
  - INCR N 64-bit or smaller writes with N=1-4 for combined individual no-write allocate stores or if in no Write-Allocate mode.
- For Normal, Non-cacheable memory:
  - INCR N 64-bit reads with N=1-4 for load multiplies and vector loads.
  - INCR N 64-bit writes with N=1-4 for combined individual stores and store multiples.
  - INCR N 64-bit reads with N=1-4 for instruction fetches.
  - INCR 1 reads of any size, for individual loads.
- For Device memory:
  - INCR 1 32-bit reads for individual load and load multiples.
  - INCR N 32-bit writes with N=1-2 for store multiple, store doubles, and vector stores.
  - INCR 1 8-bit, 16-bit reads and writes for individual subword loads and stores.
  - INCR 1 reads of any size, for vector loads.
  - INCR 1 64-bit writes for vector stores.
- INCR 1 8-bit, 16-bit, and 32-bit exclusive reads and writes for shared exclusives.
- No FIXED bursts are used.
- Write bursts to Normal memory can use the following optimizations that are allowed on AXI but have implications for bridging to AHB.
  - Entire beats with no strobes set.
  - Non-contiguous strobes per beat.

Note

- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- WRAP is a wrapping burst that is similar to an incrementing burst, except the address wraps around to a lower address if an upper address limit is reached.
- FIXED bursts, which are not used, have the same address for every transfer in the burst.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- For more information on burst types, see the *AMBA® AXI and ACE Protocol Specification*.

10.4.1.2 Data prefetching

In the high performance *Master-AXI* (M-AXI) configuration, the Cortex®-M85 processor looks at linefill addresses for L1 data cache misses. It does this to identify patterns that indicate a data stream that the software is accessing.

The data prefetcher uses the pattern information to predict where linefills might be required. It also attempts to fetch the data from the system into the L1 data cache before the data is required. This feature improves the overall performance of the processor by hiding load latency from the instructions that are executing on the processor.

The prefetcher supports two configurations: stream mode and next-line mode. In stream mode, the prefetcher can only detect streams with a constant stride. Only strides of -2, -1, +1, and +2 are supported. In next-line mode, only stride of +1 is supported. To reduce area and power, a prefetch stream cannot cross a prefetch granule boundary of 8KB.

The prefetcher is controlled using the **IMPLEMETATION DEFINED** PFCR register. This register can be used to enable or disable prefectching and to tune the prefetcher to optimize performance.

For more information on how to control the prefetcher, see 5.15 PFCR, Prefetcher Control Register on page 88.

10.4.2 Area optimized M-AXI configuration

The area optimized *Master AXI* (M-AXI) configuration supports reduced buffering and minimizes the number of outstanding AXI transactions to support a low-cost memory system without the significant area impact of a L1 data cache.

The performance for this configuration is expected to be significantly lower than the configuration described in 10.4.1 High performance M-AXI configuration on page 162, and this configuration is optimized for area alone, where practical.

10.4.2.1 Area optimized configuration M-AXI attributes and transactions

The area optimized configuration is intended to be integrated into a low-cost AXI system or bridged to AHB and is suitable for connection to a low-bandwidth memory system. For example, off-chip

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

memory. The following table shows the AXI attributes and transactions that the area optimized M-AXI configuration supports.

Table 10-4: Area optimized configuration M-AXI attributes and transactions

AXI attribute	Value	Details
Write issuing capability	32	• 15 writes to Device memory. • 17 writes to Normal memory.
Read issuing capability	3	• 2 data reads. • 1 instruction fetch or instruction linefill.
Write ID capability	3	• 1 reserved for Device memory. • 1 reserved for Normal memory Non-cacheable writes and exclusive writes. • 1 reserved for Normal cacheable writes.
Read ID capability	2	• 1 reserved for Normal Non-cacheable and Device memory. • 1 reserved for instruction fetch or instruction linefill.
Combined issuing capability	35	• 32 outstanding writes. • 3 reads from data and instructions fetches.

Only a subset of all possible AXI transactions can be generated. These are:

- For Normal memory:
  - WRAP4 64-bit reads, for instruction linefills, if a L1 instruction cache is included.
  - INCR N 64-bit reads with N=1-4 for individual loads and load multiples.
  - INCR 1 8-bit, 16-bit, and 32-bit reads for individual loads.
  - INCR N 64-bit writes with N=1-4 for combined individual stores and store multiples.
  - INCR N 64-bit reads with N=1-4, for Non-cacheable instruction fetches or all instruction fetches with no L1 instruction cache.
- For Device memory:
  - INCR 1 32-bit reads for double load multiple instructions.
  - INCR 1 8-bit, 16-bit, and 32-bit reads for individual loads.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- INCR N 32-bit writes with N=1-2 for store multiple, store doubles, and vector stores.
- INCR 1 8-bit, 16-bit, and 32-bit writes for individual stores.
- INCR 1 reads of any size, for vector loads.
- INCR 1 64-bit writes for vector stores.

- INCR 1 8-bit, 16-bit, and 32-bit exclusive reads and writes for shared exclusives.
- No FIXED bursts are used.
- Write bursts to Normal memory can use the following optimizations that are allowed on AXI but have implications for bridging to AHB.
  - Entire beats with no strobes set.
  - Non-contiguous strobes per beat.

Note

- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- WRAP is a wrapping burst that is similar to an incrementing burst, except the address wraps around to a lower address if an upper address limit is reached.
- FIXED bursts, which are not used, have the same address for every transfer in the burst.
- For more information on burst types, see the *AMBA® AXI and ACE Protocol Specification*.

10.4.3 Bridging to AHB

The high performance *Master AXI* (M-AXI) configuration is optimized for a native AXI system and not for AHB. The AHB protocol only allows one outstanding transaction. Therefore, this implies serialization of all outstanding transactions that the M-AXI can support. For acceptable levels of performance, Arm recommends that at least two AHB interfaces are used in this configuration, one for instructions and one for data.

The area optimized M-AXI configuration can be bridged to a single AHB interface if the resulting performance is acceptable.

Both M-AXI configurations support the following features that need special consideration when bridging to AHB:

Sparse write strobes

AHB does not support write strobes and therefore must split AXI beats with sparse write strobes into smaller AHB transactions. This implies that AHB write bursts can be used only when the bridge is capable of buffering an entire AXI burst and evaluating the strobes before deciding how to perform the AHB access.

To avoid this issue, the processor provides a sparse write strobe signal. Transactions can use this signal to allow AXI bursts that do not use sparse strobes to be identified before all the write data is provided. Therefore, these accesses can be performed as AHB bursts efficiently. This signal is

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

guaranteed to be valid, but in some cases it might be asserted for transactions that do not have sparse strobes.

Exclusive accesses

AMBA® AHB protocols prior to AMBA® 5 AHB do not support exclusive accesses. Arm recommends all AHB infrastructure used with the Cortex®-M85 processor is based on AMBA® 5 AHB.

The Arm® CoreLink™ AXI5 to AHB5 XHB-500 bridge, which is included in the Arm Corstone-300 Foundation IP, can be used with Cortex®-M85, and also supports the sparse write strobes signal.

10.4.4 Write response

It is a requirement of the systems using the AMBA® 5 AXI protocol that the slave does not return a write response until it has received the write address.

10.4.5 Memory system implications for AXI accesses

The attributes of the memory being accessed can affect an AXI access.

The memory system can cache any cacheable Normal memory address that has either the Read-Allocate or Write-Allocate hint.

Accesses to Device memory cannot be cached and are always Outer Shareable. Any unaligned access to device memory generates an UNALIGNED UsageFault exception and therefore does not cause an AXI transfer.

Normal Non-cacheable memory can also be Outer Shareable.

Note

Memory regions marked as Non-Cacheable Normal must not be used to access read-sensitive peripherals in a system. This is because read transactions to these regions from the processor can be repeated multiple times if the originating load instruction is interrupted.

10.4.6 Master-AXI interface transfers

The *Master-AXI* (M-AXI) interface does not generate the following types of transactions:

- An AXI slave device connected to the M-AXI interface must be capable of handling every kind of transaction that the *AMBA® AXI and ACE Protocol Specification* permits, except where there is an explicit statement in this chapter that such a transaction is not generated. You must not infer any additional restrictions from the example tables given.
- Non-cacheable load instructions might not result in an AXI transfer if they forward from an internal buffer.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Non-cacheable store instructions always result in an AXI transfer, but multiple stores might get merged into one AXI transaction.
- If the processor is powered up, the buffered write response ready signals, **BREADY** is always asserted. You must not make any other assumptions about the AXI handshaking signals, except that they conform to the *AMBA® AXI and ACE Protocol Specification*.

10.4.6.1 Restrictions on AXI transfers

The *Master-AXI* (M-AXI) interface applies restrictions to the AXI transactions it generates.

These restrictions are:

- A burst never transfers more than 32 bytes.
- The burst length is never more than four transfers.
- The maximum length of a Device write burst is two transfers. Device reads are always a single transfer.
- No transaction ever crosses a 32-byte boundary in memory.
- FIXED bursts are never used.
- The write address channel always issues INCR type bursts, and never WRAP or FIXED.
- If the transfer size is 8 or 16 bits then the burst length is always one transfer.
- The transfer size is never greater than 64 bits, because it is a 64-bit AXI bus.
- Instruction fetches are always a 64-bit transfer size, and never locked or exclusive.
- Exclusive accesses are always to addresses that are aligned for the transfer size.
- Only exclusive accesses to shared memory result in exclusive accesses on the M-AXI. Exclusive accesses to non-shared memory are marked as non-exclusive accesses on the bus.

10.5 Peripheral AHB interface

The *Peripheral AHB* (P-AHB) interface is a single 32-bit wide interface that conforms to the AMBA® 5 AHB protocol. It is designed for deterministic, data-only access to fast on-chip peripherals.

10.5.1 P-AHB interface transfers

For each clock cycle, the *Peripheral AHB* (P-AHB) interface supports one aligned 32-bit access or any 8-bit or 16-bit access that can fit inside an aligned 32-bit access. Unaligned accesses that cross a 32-bit boundary are split into multiple accesses.

Memory region type

By default, the memory regions mapped to the P-AHB interface are Device, however, it is possible to map regions as Normal using the *Memory Protection Unit* (MPU). Although Normal memory is supported on the P-AHB interface, Normal memory-specific optimization

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

is not allowed. This implies that the interface is generally unsuitable for high-bandwidth requirements, and for such a requirement, the *Tightly Coupled Memory* (TCM) or *Master AXI* (M-AXI) interface must be used instead.

Unaligned request support

The P-AHB can accommodate unaligned requests to Normal memory by breaking down the request into a set of aligned transactions that is suitable for its protocol. In most cases, the number of accesses to complete an unaligned write is greater than an equivalent read because if required, Normal memory can be excessively read, but the P-AHB interface does not support partial writes. Table 10-5: Unaligned memory access timing on page 170 lists the number of individual read and write transactions that are generated for the unaligned transactions.

Instruction execution and vector fetches support

Instruction execution and vector fetches are not supported on this interface. The P-AHB is targeted at on-chip peripherals only. Instruction and vector fetches to P-AHB are sent on the M-AXI interface.

Transactions supported

New transactions cannot be started on the bus until all outstanding transactions are completed. This implies all transactions to this interface are in-order. Loads can only start on the bus after all buffered writes are drained.

The P-AHB does not support burst transactions. This implies that, the P-AHB interface only uses one transfer and all bursts are single.

The P-AHB does not support Speculative accesses, write merging, and forwarding of buffered store data for reads. No transaction ever crosses a 4-byte boundary in memory. The transfer type is never SEQUENTIAL.

Exclusive accesses are supported in the P-AHB interface, and these accesses are always to addresses that are aligned for the transfer size. Exclusive transactions are only generated for Shareable memory regions.

The P-AHB interface can also break down sparse reads and writes that are associated with the *M-profile Vector Extension* (MVE) Load and Store instructions.

Multiple write transactions can be buffered more than once, therefore, more than one imprecise BusFault exception can be raised because of external errors. The exceptions are always raised in the same order of the store instructions which generated the transactions.

The following table assumes that only non-MVE write accesses are considered. For unaligned MVE writes, the number of accesses changes depending on the element size and predicate mask. For more information, see the *Arm®v8-M Architecture Reference Manual*.

Table 10-5: Unaligned memory access timing

Access size	Address offset	Number of read accesses	Number of write accesses
Word	+1	2	3
	+2	2	2
	+3	2	3
Halfword	+1	1	2
	+3	2	2

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

Arm recommends that the P-AHB is reserved for low-latency peripherals and all others are integrated on the M-AXI interface. This allows:

- Better overall processor execution performance in the presence of frequent stores to high-latency peripherals.
- Better *Quality of Service* (QoS) to P-AHB peripherals in interrupt handlers that do not make frequent accesses to high-latency peripherals on the M-AXI.

10.5.2 P-AHB interface configuration

The *Peripheral AHB* (P-AHB) interface covers two ranges in the processor memory map, that is, the Peripheral region and the Vendor_SYS region.

Peripheral region

Base address is fixed at `0x40000000`. The P-AHB region starts at the base address and has a size determined by PAHBCR.SZ, which is configured using the input signal **CFGPAHBSZ**.

Vendor_SYS region

The address range is `0xE0100000-0xFFFFFFFF`.

Mapping the Vendor_SYS region of the memory map to the P-AHB interface allows existing AHB-based peripherals designed for M-profile systems to be reused in Cortex®-M85-based designs.

Mapping the Vendor_SYS region of the memory map to the P-AHB interface provides additional, always-enabled, address space for direct connection to AHB-based slaves, for example, re-used peripherals from existing Cortex®-M systems.

The following parameters can be controlled for the P-AHB:

Size

The external input signal **CFGPAHBSZ** controls the size of the Peripheral region mapped to the P-AHB interface. This signal can only be changed at Cold reset. A maximum of 0.5GB is supported. This implies that the P-AHB interface is present entirely in the Peripheral region and can cover it completely. The Vendor_SYS region size is not configurable.

Enable

The external input signal **INITPAHBEN** controls the P-AHB enable state at reset. During runtime, the P-AHB Peripheral region can be enabled and disabled using the PAHBCR register. Only privileged software can modify this register.

Also, if AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. The Vendor_SYS region is always enabled.

Alias

The P-AHB interface supports the ability to alias two logical addresses in the Peripheral region onto the P-AHB interface. This feature is used with an external security gate to support fine-grain Secure and Non-secure regions in the Peripheral region. The alias bit in the logical address can be

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

configured from bit[24] to bit[28] using the external input signal **CFGMEMALIAS**. This signal can only be changed at reset.

Data accesses to the P-AHB Peripheral region are performed on the *Master AXI* (M-AXI) interface when the P-AHB interface is disabled. Accesses to the Peripheral region above the P-AHB size limit is also performed on the M-AXI interface.

Instruction accesses made to the Peripheral region, where executable, are always performed on the M-AXI interface. For code portability, Arm recommends that the P-AHB region is programmed as *Execute Never* (XN) in the *Memory Protection Unit* (MPU) to prevent instruction execution. This is consistent with the default memory map. The Vendor_SYS region is permanently XN.

10.5.3 P-AHB considerations

Normal memory is supported on the *Peripheral AHB* (P-AHB) interface. However, no Normal-specific optimizations are made. This means the interface is generally not suitable for high-bandwidth requirements, and the *Tightly Coupled Memory* (TCM) or *Master AXI* (M-AXI) interfaces must be used instead.

Instruction execution and vector fetches are not supported on this interface. The P-AHB is targeted at on-chip peripherals only.

The amount of buffering resource is intentionally limited to provide a balance between load access latency and store throughput. The implications of this limited buffering are:

- Individual stores to the P-AHB interface are visible to the Device memory in minimal and deterministic time relative to the store instruction being executed. This is relevant, for example, when an interrupt handler must perform a critical device access.
- There is limited hiding of store latency from the pipeline. This means that high-latency peripherals can stall the pipeline on a store instruction for extended periods of time. However, it affects the overall processor execution performance.
- Loads to the P-AHB interface are inherently higher latency than stores and must wait for all buffered stores to drain before they can be started on the bus. The limited buffering means that this latency is minimized but can still be significant for high-latency peripherals. The pipeline cannot flush a load that has started on the bus. Therefore, interrupt latency is affected by wait-states on loads. However, loads that have not yet started on the bus can be safely flushed. Therefore, the impact of load wait-states on interrupt latency is limited to the wait-states on a single access.
- Load access throughput is limited. There is no support for bursts on load multiples and no support for pipelined loads in general.
- Store throughput is acceptable for zero wait state systems, but it is degraded when wait states are used.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.6 S-AHB interface

The 64-bit *Slave-AHB* (S-AHB) interface provides system access to the *Tightly Coupled Memories* (TCMs). Typically, a *Direct Memory Access* (DMA) controller uses this interface to transfer data in and out of the processor for software computation. It includes arbitration logic to support simultaneous system and processor TCM access requests. The S-AHB interface implements the AMBA® 5 AHB protocol.

If there is no contention with software access to TCM and the TCM uses zero wait states, then write buffering and the read prefetcher allows the S-AHB interface to indefinitely sustain back-to-back write and read transactions.

Write buffering

Writes are buffered in the S-AHB interface to improve system performance and to provide storage for splitting 64-bit writes into two separate 32-bit transactions to the TCM interfaces.

Read access latency is inherently larger than write access latency because the AHB interface can only support a single outstanding transaction. To minimize this latency, reads can overtake buffered writes. However, if there is a data dependency between a read and a buffered write, then hazarding logic stalls the read and attempts to drain the buffer until there are no longer any dependencies. Writes are always carried out in-order and hazarding is performed at byte granularity.

Additional hazarding is included to fully serialize read accesses to the TCM from the S-AHB interface and software running on the processor. This allows both masters to access the TCM coherently. For more information on how data can be shared between software running on the processor and system-level devices that are connected on the S-AHB interface, see 10.8.5 System access to TCM through the S-AHB DMA interface on page 181.

Read prefetcher

The S-AHB interface also supports a read prefetcher to improve the performance of the processor while reading bursts of data from the TCM to the system. The prefetcher supports the following 64-bit and 32-bit read transfers:

- INCR.
- INCR4.
- INCR8.
- INCR16.

If there is no contention or wait states on the TCM banks being accessed, the prefetcher generates internal transactions so that read data can be returned on consecutive clock cycles on the S-AHB interface.

Note

- The S-AHB interface supports an extension to the AHB5 protocol using byte-lane strobe signals to efficiently handle data with non-contiguous write-data in a beat, similar to that supported on AMBA AXI interfaces. This allows for efficient bridging from an AXI-based DMA controller.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- All S-AHB accesses are treated as being the same endianness as memory. No data swizzling is performed for reads or writes.
- The S-AHB interface can be used even if the processor is in sleep mode.

10.6.1 S-AHB memory map

The memory map that is presented on the *Slave-AHB* (S-AHB) interface is consistent with the memory map that is presented to software running on the processor. Only the *Tightly Coupled Memory* (TCM) address range can be accessed. Any other addresses cause an AHB fault response.

The following table shows the S-AHB memory map.

Table 10-6: S-AHB memory map

Start address	End address	Bits [3:2] on the system address bus, HADDRS[3:2]	TCM accesses	TCM index
`0x00000000`	`0x00000000`+ ITCM size	-	ITCM	**HADDRS[n:2]**[4]
`0x20000000`	`0x20000000`+ DTCM size	00	D0TCM	**HADDRS[n:4]**
`0x20000000`	`0x20000000`+ DTCM size	01	D1TCM	**HADDRS[n:4]**
`0x20000000`	`0x20000000`+ DTCM size	10	D2TCM	**HADDRS[n:4]**
`0x20000000`	`0x20000000`+ DTCM size	11	D3TCM	**HADDRS[n:4]**

- A read or write request on the S-AHB interface to the SRAM region is mapped to 32-bit accesses to two separate DTCM instances according to **HADDRS[3:2]**.
- The processor downsizes 64-bit S-AHB accesses to the CODE region into 32 bits for ITCM accesses. A 64-bit S-AHB write transfer to ITCM are converted into two individual 32-bit buffered writes to ITCM and 64-bit S-AHB reads are converted into two ITCM serial reads that are combined into one 64-bit value for transferring over the S-AHB interface.

Note

- The TCM enable fields that are defined in the TCM control registers, ITCMCR and DTCMCR, do not affect S-AHB accesses.
- If Security gating is enabled on the TCM interface, the address ranges are aliased in the same manner as defined for software access.

10.6.2 S-AHB transfers

The *Slave AHB* (S-AHB) interface has certain conditions that require consideration.

The Cortex®-M85 processor does not support S-AHB transactions that are directly dependent on software memory transactions. This means that the system must not introduce any dependencies which imply that a software memory access cannot complete until a corresponding S-AHB transaction completes. Therefore, no loopback arrangements from processor master ports to the S-AHB interface are supported because these arrangements might cause deadlock. This

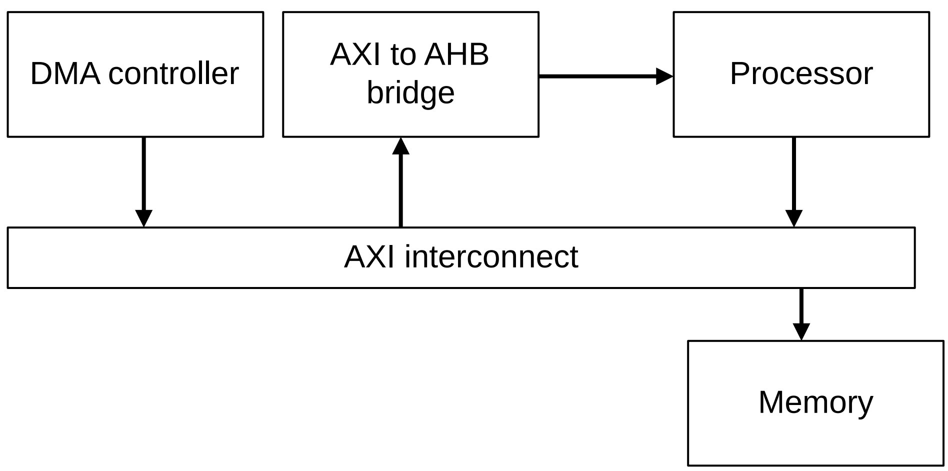
[4] The value of n depends on the configured TCM size.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

restriction does not prevent arrangements where software memory-mapped accesses are used, for example, on the *Master AXI* (M-AXI) or *Peripheral AHB* (P-AHB) interface, to request an external agent to perform transactions on the S-AHB. The only requirement is that there is no dependency introduced in the system between the control access that initiates the transaction and the transaction itself.

If a system integration contains an example design as shown in the following figure, the address decoder in th AXI interconnect that is coupled to the Cortex®-M85 processor must be configured so that the address range for the S-AHB interface is blocked to ensure that this requirement is met. The address decoder logic for other masters that are connected to the AXI interconnect are not affected by this requirement.

Figure 10-1: Example system integration



S-AHB transactions cannot perform *Memory Protection Unit* (MPU) lookups. There is no internal distinction between unprivileged and privileged S-AHB accesses. The system is entirely responsible for providing TCM protection functionality for S-AHB accesses as required. This can be carried out by performing a privilege check in either of the following areas:

- When the system memory agent has been requested for the access. This is entirely system defined and no specific hardware support is provided.
- When the S-AHB access is performed on the TCM interface. In this case, the hardware performs the TCM access at the privilege level of the S-AHB request.

- The S-AHB does not support exclusive or locked accesses and S-AHB writes do not affect the state of the internal exclusive access monitor, making it unsuitable for systems requiring concurrency controls between the S-AHB and software.

The security level for S-AHB transactions is indicated by the **HNONSECS** signal on the interface. This signal indicates the fully attributed security level. That is, after any system-level *Implementation Defined Attribution Unit* (IDAU), S-AHB accesses are not passed through or checked against the

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

processor IDAU or *Security Attribution Unit* (SAU). The TCM security gate can be used to control access to the TCM based on the transaction security level.

Note

- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- For more information on burst types, see the *Arm® AMBA® 5 AHB Protocol Specification*.

For more information on TCM security gating, see 9.8 TCM and P-AHB security access control on page 146.

10.6.3 S-AHB interface arbitration

In normal operation, there is enough bandwidth across the four *Data Tightly Coupled Memory* (DTCM) interfaces to allow accesses from software and the *Slave AHB* (S-AHB) interface to sustain their maximum throughput and the *Instruction Tightly Coupled Memory* (ITCM) is normally only used for instruction fetch. This means contention for resource should be rare and so the S-AHB is usually the lowest priority with no impact on the performance of data transfer from the system to the TCM.

However, there might be cases when a source makes large numbers of accesses to the same TCM bank. To prevent the S-AHB interface from getting less bandwidth, the priority of a request on the interface is automatically boosted when there is contention with a software access. When this occurs, a round robin scheme is used to share the bandwidth to a TCM bank roughly equally between S-AHB accesses and software accesses. This also allows the TCM bandwidth to be split evenly between software and S-AHB transactions if contention occurs.

10.6.4 S-AHB availability and low power states

The following conditions are required for the S-AHB to accept transactions:

- The processor power domain (PDCORE) is active and not in reset.
- **CLKIN** is running.

The S-AHB sub-system and the TCMs are in a separate internal clock domain to the rest of the processor. However, they are in the same reset and power domains. Therefore, S-AHB transactions can be performed without the main internal processor clock running. This allows TCM data transfers to be offloaded to a low-power system agent while the processor is in any of its sleep modes. The TCM clock is gated inside the processor to minimize the power used when no transactions are in progress from either the processor or S-AHB. Asserting **HTRANSS** automatically starts the clock if it is gated and the clock is stopped after all outstanding transactions have completed. For more information on **HTRANSS**, see C.10 S-AHB interface signals on page 353

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

From a system perspective, you are responsible to ensure that **CLKIN** is running when a transaction is started on S-AHB by considering the requirements of any master components which can access the slave interface, for example a DMA, and enabling system level clock gating accordingly. This might mean overriding the current **CLKINQACTIVE** state if the processor is in sleep and so not requesting **CLKIN**.

10.7 EPPB interfaces

The *External Private Peripheral Bus* (EPPB) interfaces are 32-bit AMBA® 4 APB interfaces designed for integration with CoreSight™ debug and trace components.

The Cortex®-M85 processor has two external EPPB interfaces:

- Core EPPB: for connection to EWIC
- Debug EPPB: for connection to TPIU, ETB, and MCU Rom Table as well as to external debug peripherals

These interfaces are used for data accesses to the memory region `0xE0040000-0xE00FEFFF`. Instruction accesses to this region cause a fault, and are permanently disabled in the Arm®v8.1-M architecture.

These interfaces are not intended for general peripheral usage, and each has both higher latency and lower average throughput than the *Master AXI* (M-AXI) or *Peripheral AHB* (P-AHB) interfaces. Additionally, they have the following limitations that make them unsuitable for general-purpose use:

- Only little-endian accesses are supported. This indicates that the processor endianness is ignored.
- All accesses are treated as Device transactions.
- Only aligned accesses are supported. Unaligned accesses to the EPPB interface cause an UNALIGNED UsageFault.
- Exclusive accesses are not supported.
- Only Privileged accesses are supported. Unprivileged accesses take a BusFault exception.

Arm recommends that all non-debug peripherals are integrated on the M-AXI or P-AHB interface.

The Debug EPPB interface can perform debugger-initiated transactions during processor reset. These transactions require the PDDEBUG power domain to be active.

For more information, see 15.2.4 Debug during reset and before code execution commences on page 240.

Additionally, for more information on EPPB peripherals, see 8.3 Private Peripheral Bus on page 134.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The Core EPPB interface is also used to transfer *Nested Vectored Interrupt Controller* (NVIC) state to an *External Wakeup Interrupt Controller* (EWIC) on sleep entry and exit. For more information on EWIC sleep entry and exit, see the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. Transactions initiated in Core EPPB require the PDCORE power domain to be active.

Note

The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document that is only available to Cortex®-M85 processor IP licensees and Arm partners with an NDA agreement.

10.8 TCM interfaces

The *Tightly Coupled Memory* (TCM) interfaces are tightly coupled into the processor for optimum performance from fast on-chip memory.

The Cortex®-M85 processor supports two separate interface groups:

ITCM

Single 64-bit interface that is intended for instruction memory based on SRAM or potentially flash memory with system prefetch or acceleration.

DTCM

Four 32-bit interfaces intended for use with data memory that is expected to be based on SRAM. The Cortex®-M85 processor performs address filtering that is based on bits[3:2] of the address.

- Addresses with bit[3:2]=`0b00` are performed on the D0TCM interface.
- Addresses with bit[3:2]=`0b01` are performed on the D1TCM interface.
- Addresses with bit[3:2]=`0b10` are performed on the D2TCM interface.
- Addresses with bit[3:2]=`0b11` are performed on the D3TCM interface.

This configuration requires that the DTCM RAM is logically arranged into four separate address banks. This allows:

- Up to 128 bits of total bandwidth for software reads and writes, and *Direct Memory Access* (DMA) traffic through the *Slave AHB* (S-AHB) interface with a probabilistic reduction of contention. This is essential for compute performance because the Cortex®-M85 processor can sustain a data throughput of 64 bits per cycle using the *M-class Vector Extension* (MVE) instructions.
- A 64-bit bandwidth for contiguous accesses that are 32-bit aligned from the software and DMA. A 64-bit bandwidth for contiguous accesses is essential for both overall performance and interrupt latency.
- Dual-issuing of 32-bit (or lower) read transactions, where the two addresses do not contend.
- Dual-issuing of 32-bit (or lower) write transactions, where the two addresses do not contend.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Dual-issuing of 64-bit (or lower) aligned read transactions with 64-bit (or lower) aligned write transactions to the DTCM from software, where the two addresses do not contend.

Note

- The Cortex®-M85 processor does not provide software control over address filtering.
- All TCM interfaces support wait and error response from external memory. For systems where functional safety or *Reliability, Availability, and Serviceability* (RAS) are required, the Cortex®-M85 processor also optionally supports a *Single Error Correction and Double Error Detection* (SECDED) scheme that is based on the *Error Correcting Code* (ECC) for all accesses in the ITCM and DTCM regions.
- To configure the processor to support ECC, see the configuration options in the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is only available to licensees.

10.8.1 TCM configuration

The TCM interface has fixed and configurable parameters.

The base address of each TCM is fixed:

ITCM

`0x00000000`. This is the base address of the Code region.

DTCM

`0x20000000`. This is the base address of the SRAM region.

The following parameters can be separately controlled for each of the TCMs:

Size
External configuration input signals control the size of each TCM region. These signals can only be changed at Cold reset. A maximum of 16MB for each TCM is supported. This implies that the ITCM and DTCM are present entirely in the Code and SRAM regions of the memory map respectively.

Enable
An external input signal controls the TCM enable state at reset. During runtime the TCM can be enabled and disabled using the ITCMCR and DTCMCR registers. Only privileged software can modify these registers. If AIRCR.BFHFNMINS is zero, these registers are RAZ/WI from Non-secure state.

Alias
The TCM controller can alias two logical addresses in the Code and SRAM regions onto the ITCM and DTCM interface respectively. This feature is used with the TCM security gate to support fine-grain Secure and Non-secure regions in TCM memory. The alias bit in the logical address can be configured from bit[24] to bit[28] using the external input signal **CFGMEMALIAS**. This signal can only be changed at Cold reset.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- For more information on ITCMCR and DTCMCR registers, see 5.19 ITCMCR and DTCMCR, TCM Control Registers on page 97.
- For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.
- Address aliasing and security gating are described in 9.8 TCM and P-AHB security access control on page 146.
- To configure the processor to support ECC in the TCMs, set `ECC` to TRUE. See the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document that is available only to licensees and Arm partners with an NDA agreement.

10.8.2 TCM transactions

TCM regions are implicitly Normal, Non-shareable, Non-cacheable memory.

For TCM memory regions, the Cortex®-M85 processor:

- Ignores the *Memory Protection Unit* (MPU) memory type attributes that software assigns. The MPU protection settings are always considered.
- Initiates Speculative reads. You must not assume that the scope of this speculation is fixed, or that it can be definitively specified. For example, speculation might occur:
    - For instruction prefetching, depending on the recent execution stream.
    - For data reads that are performed before the Security or MPU protection settings are evaluated. Although the access might be performed speculatively, an abort is subsequently raised if required by the Security or MPU protection settings.
    - For data reads in branch shadows.
- Buffers data on writes. Read transactions always hazard against outstanding buffered write transactions to the same address. Writes transactions are never Speculative.

This behavior makes TCMs unsuitable for peripherals or any memory that has implications for read transactions. Devices of this type must be integrated on the *Peripheral AHB* (P-AHB) or *Master-AXI* (M-AXI) interfaces. These interfaces support the Device memory type. Additionally, the following accesses are performed on the M-AXI interface instead of the TCM interfaces:

- Accesses to TCM regions when the relevant TCM is disabled.
- Accesses to the Code and SRAM regions above the TCM size limit, regardless of the TCM enable.

For code portability to other Arm processors or systems, Arm recommends that TCM regions are always defined as Normal, Non-shareable memory in the MPU.

This is consistent with the default memory map attributes which apply when the MPU is disabled.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.8.3 Booting from TCM

The Cortex®-M85 processor provides support for booting from volatile TCM memory that must be initialized at reset.

The TCMs can be enabled out of reset without software programming. When the **CPUWAIT** signal that stalls the core is HIGH out of reset, it prevents the processor from executing any software at the reset vector. This allows the TCMs to be loaded by the system before the processor performs any TCM accesses. When the TCM loading sequence is complete, this signal can be deasserted to allow the processor to boot up. The *Slave AHB* (S-AHB) *Direct Memory Access* (DMA) interface is functional when the **CPUWAIT** signal that stalls the core is asserted out of reset and can therefore service transactions that the system initiates to load the TCMs. This avoids the need for external hardware on the TCM interface for boot-time initialization.

Note

Asserting **CPUWAIT** prevents the processor from reading the stack pointer (SP) or initial program counter (PC) from the reset vector. Therefore, it is safe to load the vector table, code, and data into the TCM. Alternatively, the external input signals **INITSVTOR** and **INITNSVTOR** can be used to set the vector table address in non-volatile memory.

When ECC is enabled, before performing a byte, halfword, or unaligned word write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned word or doubleword write to the location. Arm recommends that all TCM locations are initialized in this manner by boot code.

10.8.4 Integration with flash memory

The Cortex®-M85 processor can support the use of flash memory connected to *Tightly Coupled Memory* (TCM). The *Instruction Tightly Coupled Memory* (ITCM) interface is most suitable for this arrangement.

The system must take into account the fetch bandwidth requirements for efficient code execution by the processor. The processor can consume up to 32 bits of instruction data per cycle using both 32-bit Thumb and 16-bit Thumb instructions, because the 16-bit Thumb instructions can be dual-issued. The overall bandwidth is specific to your application but for general-purpose products, it must be assumed that 32 bits per cycle might be required. The instruction memory system needs to sustain this for maximum performance. Arm recommends that if flash memory is integrated on the ITCM, some system cache or Flash accelerator is used to meet these fetch bandwidth requirements.

Alternatively, flash memory can be integrated on the *Master AXI* (M-AXI) and the processor can be configured to include an L1 instruction cache.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.8.5 System access to TCM through the S-AHB DMA interface

The 64-bit *Slave-AHB* (S-AHB) interface provides system access to the *Tightly Coupled Memory* (TCM) even when the Cortex®-M85 processor is running.

Typically, this feature is used with a *Direct Memory Access* (DMA) controller to transfer data to and from the processor for compute applications. Arbitration between processor access from software and S-AHB requests to TCM is fully supported with no requirement for external TCM interface logic. For more information on this arbitration logic, see 10.6.3 S-AHB interface arbitration on page 176.

There is no hardware support for concurrency control between software and S-AHB access to TCM. Particularly, software exclusive accesses to TCM are only subject to the internal exclusive monitor which does not take S-AHB accesses into consideration. This implies that the system must not perform S-AHB accesses to any regions of TCM memory that are used with software exclusive accesses. However, it is possible in software to share data coherently between the executing thread and the S-AHB interface. The processor makes the following hardware guarantees to share data coherently:

- Appropriate writes to the TCM by software and S-AHB are never repeated. Store double instructions, floating-point store multiple instructions storing double-precision values, *M-profile Vector Extension* (MVE) stores, and unaligned single stores can be repeated on exception return. Therefore, these transactions are exempt from this guarantee and unsuitable for software synchronization. The processor guarantees that no single-copy-atomic access is repeated.
- Software and S-AHB writes to the TCM have a single point of serialization which is the *TCM Control Unit* (TCU). This means that when a write is observable by one master, it is guaranteed to be observable by the other.
- When a write on the S-AHB interface is accepted, the processor assumes responsibility for the coherent observation of that data. Any read by any master interface that is initiated after the S-AHB write completed returns the updated data.

Note

- TCMs are implicitly Normal memory, therefore, write buffering is permitted.
- All TCU buffers are drained before the processor enters a low-power sleep state.

The following table shows an example software sequence for message passing between coherent components in a system.

Table 10-7: Example software sequence for message passing between coherent components in a system

Data generator	Data consumer
`STR` <data>	`LDR` <valid>
`STL` <valid> : Store-release	`LOOP` until <valid> set
	`LDA` <data> : Load acquire

The S-AHB interface always performs writes in-order, and therefore, it does not need a barrier when generating data into the TCM.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Interrupt-based synchronization is also possible in the Cortex®-M85 processor when the S-AHB is the data generator. In this model, an interrupt is generated when the last data transfer completes on the external interface. The first instruction in the *Interrupt Service Routine* (ISR) is guaranteed to observe any data items that are stored before or on this transfer. In this case, the completion of the last S-AHB access is used to indicate global observability instead of performing a software read of the location and waiting until it has been updated.

For more information on the S-AHB interface, see 10.6 S-AHB interface on page 172.

10.9 Instruction and data cache

The Cortex®-M85 processor supports optional, internal L1 Harvard caches for high performance operation using on-chip or external memory.

Only the *Master-AXI* (M-AXI) interface accesses can be cached. TCM and *Peripheral AHB* (P-AHB) interface transactions or accesses cannot be cached.

To enable software to appropriately deal with different levels of cache, the cache maintenance operations can perform up to the following points:

Point of Unification **(PoU)**

This is the point at which the instruction and data caches can see the same copy of a memory location. For the Cortex®-M85 processor:

- When either an L1 data cache or an instruction cache is included, the PoU is always at the system level, therefore, cache maintenance operations by address always act on the L1 cache. This is indicated by CLIDR.LoUU and CLIDR.LoUIS bitfields. This implies that the data and instruction cache accesses are unified at the system level.
- When the data cache and instruction cache are excluded, the CLIDR.LoUU and CLIDR.LoUIS bits are `0b000`.

Point of Coherency **(PoC)**

This is the point at which all components that can access memory can see the same copy of a memory location. For the Cortex®-M85 processor:

- When either an L1 data cache or instruction cache is included, the PoC is always at the system level, therefore, cache maintenance operations by address always act on the L1 cache. This is indicated by CLIDR.LoC bit field. This implies that data accesses are coherent at the system level or beyond the system level.
- When the L1 data cache and instruction cache are excluded, the CLIDR.LoC bit is `0b000`.

For more information on the CLIDR register, see 5.5.1 CLIDR, Cache Level ID Register on page 60. Each cache can be independently configured within the following range:

- 4KB
- 8KB
- 16KB
- 32KB

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- 64KB

Both the L1 instruction cache and data caches store the valid bits for each cache line in RAM. The Cortex®-M85 processor provides a hardware mechanism to invalidate the cache at reset. This mechanism can be disabled to maintain valid cache state across reset, for example, where the RAM supports data retention and the processor logic is reset after powerup.

The automatic invalidation sequence can take a large number of cycles and executes independently of the instructions that are running on the processor. While the automatic invalidation sequence is in progress, any cache maintenance operation is treated as a NOP and instructions and data accesses do not look up in the cache. A `DSB` instruction waits for all automatic cache invalidate sequences to complete.

Software can also be used to perform a complete invalidation before enabling the data cache on reset. The L1 instruction cache can be invalidated by a single instruction but the L1 data cache needs a loop iterating through all entries.

The architecture specifies the cache maintenance operations which can be used by software. The Cortex®-M85 processor includes memory-mapped registers that allow software to examine the content of the cache tag and data RAMs directly. This can be used for profiling or debugging the cache content. See 5.11 Direct cache access registers on page 71 for more information. The Direct Cache Access registers are only accessible in Secure state. Therefore, there is no requirement to restrict cache readability. The processor supports direct access to the cache RAM, therefore, access to the L1 instruction cache must also be restricted. This can be achieved by asserting the external input signal **LOCKDCAIC**. For more information on **LOCKDCAIC**, see C.28 Miscellaneous signals on page 368.

Dirty data must be written back to external memory before the processor and RAM are powered down because the L1 data cache supports write-back operation.

All cache RAMs are standard single-ported RAMs and can be generated using standard RAM compilers.

10.9.1 L1 data cache

The Cortex®-M85 processor L1 data cache has the following features:

- It is a four-way set-associative cache.
- It has a cache line size of 32 bytes.
- It supports the following inner memory attributes and allocation hints for Non-shareable memory:
    - Write-Back and Write-Through Cacheable.
    - Read-Allocate and No Read-Allocate.
    - Write-Allocate and No Write-Allocate.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Transient and Non-transient. Clean cache lines that are associated with Transient memory are prioritized for eviction over lines that are associated with Non-transient memory.

  Allocation into the L1 data cache depends on inner memory attributes only.

- The outer and inner memory attributes are exported on the *Master AXI* (M-AXI) interface to support further system-level caching.
- The Shareability attribute forces the region to be treated as Non-cacheable, regardless of the inner memory attributes. This enables maintaining coherency at the system-level.

Software or a debugger might use the direct cache access registers to read the contents of RAM arrays. The data cache is logically organized into two sets of RAM arrays. The dimensions of these RAM arrays vary with the cache size and the inclusion of *Error Correcting Code* (ECC) logic.

Table 10-8: Data cache RAM organization

Array	Number of cache instances	Data stored	Write granularity	Array width excluding ECC (bits)		Array width including ECC (bits)		Array depth (number of entries)	
				4KB	64KB	4KB	64KB	4KB	64KB
Tag	4	Tag, valid, line status	RAM word	27	23	33	29	32	512
Data	8	Data	Byte	32	32	39	39	128	2048

10.9.1.1 No Write-Allocate mode

When a memory region is marked as Cacheable Write-Allocate, it normally allocates a cache line on a write miss. However, there are some situations where allocating on writes is undesirable, such as executing the C standard library `memset()` function to clear a large block of memory to a known value.

Writing large blocks of data like this can pollute the cache with unnecessary data. It can also waste power and performance if a linefill must be performed only to discard the linefill data because the entire line was subsequently written by the `memset()`.

To prevent this, the Cortex®-M85 data cache includes logic to automatically disable data cache allocation on a write miss when streaming behavior is detected. When in this mode, writes are buffered and then written directly out to the external system through the *Master-AXI* (M-AXI) interface even if they are cacheable.

No Write-Allocate mode is enabled when the data cache detects that three consecutive linefills have been overwritten by write data before being allocated to the cache. When enabled, the processor remains in No Write-Allocate mode until either:

- A linefill is allocated where a store has not overwritten a read from the M-AXI interface.
- A linefill is started on an address which hazards on a buffered write or an outstanding write to the M-AXI interface, indicating that it is unlikely to be related to the write data stream.

No Write-Allocate mode can be disabled by setting the ACTLR.DISNWAMODE to 1.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For more information on ACTLR, see 5.8 ACTLR, Auxiliary Control Register on page 65 .

10.9.2 L1 instruction cache

The Cortex®-M85 processor L1 instruction cache has the following features.

- It is a two-way set-associative cache.
- It has a cache line size of 32 bytes.
- It does not allow writes to be performed, except for allocations.
- It only supports Read-Allocate for Inner Cacheable memory. Write-Allocate, Write-Back, Write-Through, and Transient attribute hints are ignored. Allocation into the L1 data cache depends on inner memory attributes only.
- Outer and inner memory attributes are exported on the *Master-AXI* (M-AXI) interface to support further system-level caching.
- The Shareability attribute is ignored for instruction side accesses.
- The Inner Cacheability attributes are always respected.

Debug accesses from the *Debug AHB* (D-AHB) slave interface on the processor cannot read information from the instruction cache.

Software or a debugger must use the direct cache access registers to read the contents of RAM arrays. The instruction cache is logically organized into two sets of RAM arrays. The dimensions of these RAM arrays vary with the cache size and the inclusion of *Error Correcting Code* (ECC) logic.

Table 10-9: Instruction cache RAM organization

Array	Number of cache instances	Data stored	Write granularity	Array width excluding ECC (bits)		Array width including ECC (bits)		Array depth (number of entries)	
				4KB	64KB	4KB	64KB	4KB	64KB
Tag	2	Tag and valid	RAM word	22	18	28	24	64	1024
Data	2	Instructions	RAM word	64	64	71	71	256	4096

10.9.3 Cache maintenance operations

All cache maintenance operations are performed through word stores to the *Private Peripheral Bus* (PPB) space using the relevant PPB architectural registers.

The following table lists the cache maintenance operations that are associated with the relevant cache type.

Table 10-10: Cache maintenance operations

Operation	L1 cache type	Register
Invalidate all	Instruction cache	ICIALLU

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Operation	L1 cache type	Register
Invalidate by address	Instruction cache and data cache	ICIMVAU, DCIMVAC
Invalidate by set/way	Data cache only	DCISW
Clean by address	Data cache only	DCCMVAU, DCCMVAC
Clean by set/way	Data cache only	DCCSW
Clean and invalidate by address	Data cache only	DCCIMVAC
Clean and invalidate by set/way	Data cache only	DCCISW

Cache maintenance operations require software to use barriers carefully to guarantee intended operation:

- A `DMB` instruction is required to guarantee that a cache maintenance operation does not affect previous memory accesses.
- A `DSB` instruction is required to guarantee completion of all outstanding cache maintenance operations and to guarantee that outstanding cache maintenance operations do not affect any subsequent memory accesses.
- An `ISB` instruction is required to guarantee that the effects of all completed cache maintenance operations are visible to subsequent instruction fetches.

For more information on these barrier instructions, see the *Arm®v8-M Architecture Reference Manual*.

Cache maintenance is required when changing security attribution of an address by either reprogramming the *Security Attribution Unit* (SAU) or changing the external *Implementation Defined Attribution Unit* (IDAU) mappings.

Cache maintenance operations are supported in both Secure and Non-secure state. Software operating in Non-secure state cannot change secure data. Therefore, the behavior of some operations in Non-secure state is:

- Data Cache Line Invalidate by Set/Way (DCISW) is promoted to Data Cache Line Clean and Invalidate by Set/Way (DCCISW)
- Data Cache Line Invalidate by Address to *Point of Coherency* (PoC) (DCIMVAC) and Data Cache Line Invalidate to *Point of Unification* (PoU) are both promoted to Clean and Invalidate the data cache line which includes the selected address.

The Non-secure invalidate operations are only promoted if the processor is configured with the Secure extension.

There are no data cache maintenance operations that operate on the entire cache. However, the processor provides a mechanism to automatically invalidate the cache at reset to initialize the structure before use.

Software can implement operations across the entire data cache by using the set/way operations to iterate across all the sets and ways of the cache.

For more information on cache maintenance operations, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.9.4 Automatic cache invalidation at reset

If the L1 caches move from an unpowered to a powered state, the caches are automatically invalidated. Automatic invalidation is also initiated when the RAM power domain is powered up when the core power domain is already active. For example, if the cache is re-enabled after it was shutdown to save power when not in use.

A small counter starts at the bottom of the caches and invalidates one line at a time. Until the automatic invalidation completes, any cache maintenance operation is treated as a NOP, no cache lookup or allocate is performed, and all data accesses to Normal Cacheable memory are effectively treated as Non-cacheable.

The automatic invalidation does not occur on transition to, or from, a cache retention state when controlled by the P-Channel interface. Automatic cache invalidation at reset can be disabled through the **INITL1RSTDIS** top-level input signal. Tying **INITL1RSTDIS** to 1, allows cache state to be maintained across reset. This can be used when the processor integration does not support power control using the P-Channel interface and the cache RAM supports state retention.

The invalidation sequence executes independently of the instructions running on the processor and is significantly more efficient that the equivalent software sequence. The instruction and data cache are invalidated in parallel with all cache ways invalidated simultaneously (two instruction cache lines and four data cache lines per cycle).

Note

- While the automatic invalidation sequence is in progress, any cache maintenance operation is treated as a NOP and instruction and data accesses do not look up in the cache.
- If a `DSB` instruction is executed while the automatic invalidation sequence is in progress the instruction stalls the processor until the sequence is completed. The `DSB` can be interrupted if an exception of sufficient priority is pended and the automatic invalidation sequence continues. For more information on the instruction, see the *Arm®v8-M Architecture Reference Manual*.

The L1 data cache supports write-back operation. Therefore, dirty data must be written back to external memory before the processor and RAM are powered down. The processor provides register fields MSCR.DCACTIVE and MSCR.DCCLEAN to carry out this procedure.

For more information on MSCR, see 5.13 MSCR, Memory System Control Register on page 84.

10.9.5 Cache coherency

The Cortex®-M85 processor does not support hardware coherency for the L1 instruction and data caches. Coherency can only be maintained at the system level.

The following table summarizes the cache coherency usage models that the L1 data cache supports. The L1 instruction cache always follows the programmed Cacheability attributes and it is

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

unaffected by the Shareable attribute that is defined in MPU_RBAR.SH for the MPU region that is associated with an address. For more information on MPU_RBAR, see the *Arm®v8-M Architecture Reference Manual*.

Further levels of caches are also supported.

For more information on further levels of caches, see 10.9.7 System cache support on page 190.

Table 10-11: Coherency usage models available on the Cortex®-M85 processor

MPU_RBAR.SH	Scenario description for L1 data cache
`0b10`, `0b11`	• All shareable locations are treated as inner Non-cacheable. • Programmed inner Cacheability attributes are ignored. • The L1 data cache is transparent to software for these locations. Therefore, no software maintenance is required to maintain coherency.
`0b00`	• Programmed inner Cacheability attributes are considered. • Data is not shared with other agents. Therefore, coherency issues do not exist.

Note

The L1 instruction cache always considers the programmed Cacheability attributes and the Shareability attribute defined in MPU_RBAR.SH does not affect it.

10.9.6 Accessing the caches

If the Cortex®-M85 processor has been configured to include an instruction or data cache, the CCR and MSCR registers are responsible for controlling access to the caches.

The following register bits are responsible for cache access:

- CCR.DC and CCR.IC are cache enable bits for the instruction cache and data cache respectively. If these bits are set to 0, then cache allocation is not allowed. Loads and stores can lookup and hit in the cache. Cache maintenance operations and direct cache accesses work normally.
- MSCR.DCACTIVE and MSCR.ICACTIVE control cache access for the instruction cache and data cache respectively. If these bits are set to 0, then load and stores do not lookup or hit in the cache, and cache maintenance operations and direct cache accesses do not access the cache. These bits also serve as a hint to the system to indicate that power can be removed from the cache.

The following table describes the different cache access scenarios.

Table 10-12: Cache access scenarios

CCR	MSCR	Cache access behavior
CCR.DC and CCR.IC are set to 1	MSCR.DCACTIVE and MSCR.ICACTIVE are set to 1	Normal operating mode. Unless PDCORE goes OFF resulting in PDRAMS going to RET, the caches are powered up and cache accesses can perform allocation and lookup.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

CCR	MSCR	Cache access behavior
CCR.DC and CCR.IC are set to 0	MSCR.DCACTIVE and MSCR.ICACTIVE are set to 1	Cache lookups are allowed, but cache allocation is not permitted. This behavior is used to clean the cache before powering down.
CCR.DC and CCR.IC are set to 0 or 1	MSCR.DCACTIVE and MSCR.ICACTIVE are set to 0	The caches are not being used, and they can be powered down. The CCR.DC and CCR.IC bits are ignored.

Note

- For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on MSCR, see 5.13 MSCR, Memory System Control Register on page 84.
- For more information on PDCORE and PDRAMS, see 7.1 Power domains on page 115.

10.9.7 System cache support

The following table shows the two optional levels of cache that the architecture implicitly defines.

Table 10-13: System cache levels supported by Arm®v8.1-M and Cortex®-M85

Cache level	Implemented by	Controlled by
L1	Internal processor caches	Inner Cacheability attributes
System level (L2)	External L2 cache controller integrated on the *Master AXI* (M-AXI) interface.	Outer Cacheability attributes **Note:** The Outer Cacheability attributes are exported, and the L2 cache controller uses the **ARCACHE** and **AWCACHE** signals to determine these attributes. For more information on these signals see, C.9 M-AXI interface signals on page 349. The **ARINNER** and **AWINNER** signals, which define the Inner Cacheability attributes can be used as hints for the L2 cache controller to optimize allocation or caching policy. The **ARINNER** and **AWINNER** signals can be used for debugging and monitoring purposes.

10.9.8 Direct cache access

The Cortex®-M85 processor provides a mechanism to read the embedded RAM that the L1 data and instruction caches use through **IMPLEMENTATION DEFINED** system registers. This functionality is useful to investigate data coherency issues.

There are four direct cache access registers:

- The read registers, DCADCRR and DCAICRR, for the L1 data and instruction cache respectively.
- The location registers, DCADCLR and DCAICLR, for the L1 data and instruction cache respectively.

Direct cache access registers are only accessible from the Secure privileged state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- For more information on DCADCRR and DCAICRR, see 5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74.
- For more information on DCADCLR and DCAICLR, see 5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers on page 72.

Reading a cache location

To read a cache location, the following steps must be performed in order:

1. The cache location to be read is written to the appropriate location register.
2. A read is then performed to the corresponding read register. This returns the data from that cache RAM location.

The location that is specified must be a physical RAM address. The processor translates the cache way into the appropriate RAM bank. The logical cache way and the physical RAM bank can be different because of the internal organization of the cache.

Example code sequence for reading an instruction cache location

```
DCAICLR EQU 0xE001E214 ; Direct Cache Access Instruction cache Location
 Register address
DCAICRR EQU 0xE001E204 ; Direct Cache Access Instruction cache Read Register
 address

MOV R3, 0x0      ; Start building the value to write into the DCAICLR
                 ; Bit[0]==0b0, to target the tag RAM
LSL R0, #5
ORR R3, R0       ; Put the cache index into bits[14:5] of DCAICLR

LSL R1, #31
ORR R3, R1       ; Put the way into bit[31] of DCAICLR

LDR R11, =DCAICLR
STR R3, [R11]    ; Write the location into DCAICLR

LDR R11, =DCAICRR
LDR R4, [R11]    ; Read DCAICRR, R4 will be updated with the contents of the
 Instruction cache tag
                 ; at the supplied index and way
```

ECC errors

Direct accesses ignores all *Error Correcting Code* (ECC) errors and cannot be used to read the ECCs in the RAMs.

Accessing a cache location

For details on the encoding of the DCADCRR and DCAICRR registers, see 5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74.
When the data RAM is specified in either the DCADCLR[0] or DCAICLR[0], the data offset field determines the word that is read which is in DCAxCLR[5:1].

When the tag RAM is specified in DACDCLR[0] or DCAICLR[0], the tag encoding that is written to DCADCRR or DCAICRR for the data and instruction cache respectively is shown in the following tables. Unused fields in the data register are written as zero.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 10-14: DCADCRR data format for data cache tag RAM reads

Cache size	Status bits	Valid bit	Tag bits
4KB	[25:23]	[22]	[21:0]
8KB	[25:23]	[22]	[21:1]
16KB	[25:23]	[22]	[21:2]
32KB	[25:23]	[22]	[21:3]
64KB	[25:23]	[22]	[21:4]

Table 10-15: DCAICRR data format for instruction cache tag RAM reads

Cache size	Valid bit	Tag bits
4KB	[21]	[20:0]
8KB	[21]	[20:1]
16KB	[21]	[20:2]
32KB	[21]	[20:3]
64KB	[21]	[20:4]

The STATUS bits in the data cache tag RAM contain information regarding:

- The clean/dirty status.
- Arm®v8.1-M transient attribute for a valid cache line.
- Outer attributes for a valid cache line.

For more information on the STATUS bits, see 5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74.
The following table describes the information that is stored in a state-dependent format.

Table 10-16: Data cache tag RAM status encoding

Status encoding	Line Clean/Dirty	Line Transient	Outer attributes
`0b000`	Clean	Yes	UNKNOWN
`0b001`	Clean	No	UNKNOWN
`0b010`	Dirty	No	Non-cacheable
`0b011`	Dirty	No	Write-Back, Write-Allocate
`0b100`	Dirty	No	Write-Back, No Write-Allocate
`0b101`	Dirty	No	Write-Through, Write-Allocate
`0b110`	Dirty	No	Write-Through, No Write-Allocate

Note

- `0b111` is reserved.
- Outer attributes are only valid for lines allocated to Inner write-back memory regions when they are made dirty by a write.
- Only clean lines can be distinguished as transient. When a line has been written as dirty, it is evicted from the cache by a subsequent line-fill with the same priority as other non-transient lines.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.10 Store buffer

The memory system includes a *STore Buffer* (STB) to hold data before it is written to the cache RAMs or passed to the *Master-AXI* (M-AXI) interface. The STB also holds data to be written to TCMs, P-AHB, and PPB. All store instructions must pass through the STB.

- The STB has six identical slots which hold the address, up to 64 bits of data, and other attributes for AXI store transactions.
- The STB has five identical slots which hold the address, up to 64 bits of data, and other attributes for TCM and PIU store transactions.

10.10.1 Store buffer merging

The *STore Buffer* (STB) has merging capabilities. If a previous write access has updated an entry, other write accesses on the same doubleword can merge into this entry. Merging is only possible for stores to Normal memory.

Merging is not possible if:

- The access is to Device memory.
- The first access leaves the STB, either on the AXI or to the cache, before the second access reaches the STB.
- Either access is a Store-Exclusive.
- The second access is a Store-Release.

10.10.2 Store buffer behavior

The *STore Buffer* (STB) directs cacheable write requests to the cache controller and *Master-AXI* (M-AXI) interface blocks. The STB also directs requests to be written to TCMs, P-AHB, and PPB interfaces.

Cache controller for cacheable write hits

The store buffer sends a cache lookup to check that the cache hits in the specified line, and if so, the store buffer merges its data into the cache when the entry is drained

M-AXI interface

For Non-cacheable, and Cacheable No Write-Allocate stores that miss in the L1 data cache, a write access is performed on the M-AXI interface.

For Cacheable Write-Allocate stores that miss in the data cache, a linefill is started using any of the linefill descriptors. The store data is merged into the cache once the linefill tag is written to the cache. If the STB slot has an entire data for that chunk, then that slot can write to the data cache as soon as the linefill tag is written. Otherwise, STB will wait until the entire cacheline is written into the cache.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

PIU write transactions

The STB sends write transactions that target the PIU to the PIU 32-bits at a time.

TCM write transactions

The STB sends write transactions that target the TCM to the TCU and can drain up to four 32-bit write transactions per cycle.

10.10.3 Store buffer ordering

The *STore Buffer* (STB) has ordering capabilities and must maintain ordering between some stores.

The STB ordering is compulsory for the following stores:

- All Device stores must occur in order with respect to other Device accesses.
- Stores after a load-acquire must occur after the load-acquire.
- Stores before a store-release must occur before the store-release.

10.10.4 Store buffer draining

The *STore Buffer* (STB) is drained of all stores to Device memory before a load is performed from Device memory.

Slots that are Non-mergeable drain quickly because there is no benefit in being present in the STB. Mergeable slots might wait for future stores to merge into them and reduce the number of cache writes required.

A store buffer entry is drained if:

- There is a cache maintenance operation pending.
- There is a store that cannot enter the STB because of the current contents of the STB.
- There is a `DSB`, `DMB`, `ESB`, `WFI`, or `WFE` instruction.
- There are debug events.

Note

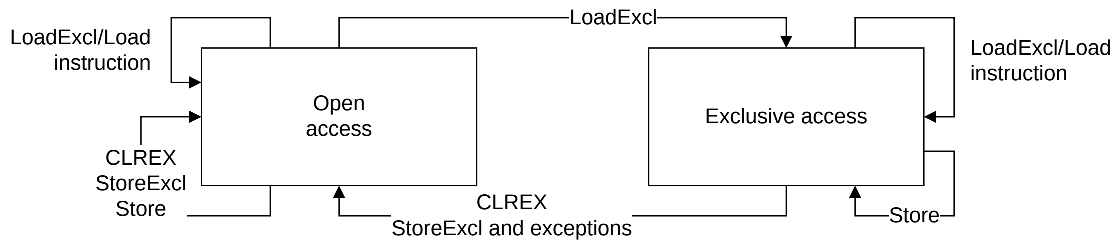
The STB can drain up to 128-bits of data to the cache in one cycle.

10.11 Internal local exclusive access monitor

The Cortex®-M85 processor implements an internal local exclusive access monitor that does not tag addresses. This implies that the reservation granule is the entire memory.

The following figure shows the operation of the internal local exclusive monitor, including all **IMPLEMENTATION DEFINED** options.

Figure 10-2: Operation of internal exclusive access monitor



- LoadExcl are exclusive load instructions to addresses associated with the *Tightly Coupled Memory* (TCM), *Master AXI* (M-AXI), and *Peripheral AHB* (P-AHB) interfaces which are either Non-shareable or Shareable when the system supports a global exclusive monitor.
- Exclusive Load instructions which access addresses in the *Private Peripheral Bus* (PPB) region, including the *Internal Private Peripheral Bus* (IPPB) registers and the *External Private Peripheral Bus* (EPPB) interface do not update the internal exclusive monitor.
- Exclusive Load instructions do not update the internal exclusive monitor if these instructions are in Shareable memory addresses associated with the M-AXI and P-AHB interfaces where a global exclusive monitor is not supported.
- Exclusive Store instructions (StoreExcl) always clear the internal exclusive monitor.
- *Slave AHB* (S-AHB) accesses to TCM do not affect the internal local exclusive access monitor. There is no hardware support for concurrency control between software and S-AHB to TCM.
- *Memory Built-In Self Test* (MBIST) and *Debug AHB* (D-AHB) accesses do not affect the internal local exclusive access monitor.
- Exception entry and return are architecturally defined to clear the local exclusive access monitor.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10.12 M-AXI and P-AHB interaction with the global exclusive monitor

The *Master AXI* (M-AXI) and *Peripheral AHB* (P-AHB) interfaces support systems that include a global exclusive monitor by using the interface signals that conform to the AMBA® 5 AXI and AMBA® 5 AHB protocols respectively.

Accesses associated with load and store exclusive instructions are only handled as exclusive on the M-AXI and P-AHB interfaces if they are either of the following:

- Device memory.
- Normal memory marked as Shareable in the associated *Memory Protection Unit* (MPU) region.

Exclusive accesses to Normal Shareable memory are always treated as Shareable Non-cacheable by the processor.

Only the internal exclusive access monitor handles accesses to Non-shareable memory.

If an Exclusive read access is carried out to a region that does not support a global exclusive monitor, the slave must respond in either of the following ways:

- An OKAY response for AXI.
- The HEXOKAYP response must be deasserted for P-AHB.

These responses do not result in the processor taking an exception, but they do ensure that the `STREX` does not pass. This kind of livelock behavior can be trapped using a Watchdog unit.

Note

The default memory map includes only Non-shareable Normal memory regions. For more information on the memory map, see 8.1 Memory map on page 132

10.13 MBIST

The Cortex®-M85 processor supports the production *Memory Built-In Self-Test* (MBIST) use model.

This allows memory testing during manufacture. This use model requires that a production MBIST controller is inserted into the processor and connected to the internal MBIST interface. This can be automatically carried out by EDA tools using configuration information that is delivered with the processor.

Note

The Cortex®-M85 processor does not support an external MBIST interface.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

11 Reliability, Availability, and Serviceability Extension support

This chapter describes the *Reliability, Availability, and Serviceability* (RAS) features implemented in the Cortex®-M85 processor.

11.1 Cortex®-M85 processor implementation of RAS

The Cortex®-M85 processor implements the Arm®v8.1-M *Reliability, Availability, and Serviceability* (RAS) features to ensure correct operation in environments where functional safety and high-availability are critical. The RAS Extension is always included in the Cortex®-M85 processor, however most of the features are only supported when *Error Correcting Code* (ECC) is configured and enabled.

The Cortex®-M85 processor standardizes the software interface for fault detection and analysis by supporting the RAS Extension. The RAS features supported are *Error Correcting Code* (ECC) for the L1 instruction cache and data cache, and TCMs.

Errors are reported to the system through:

- Output signals on the processor. For more signal information, see C.29 Error interface signals on page 372.
- Error bank registers which can be used to mitigate hard errors that cannot be corrected by writing back to the RAM. For more information, see 5.12 Error bank registers on page 78.
- The architectural registers that are defined by the RAS Extension. For more information, see 11.4 RAS Extension registers on page 207

Supported RAS architectural features

The RAS architecture contains:

- An *Error Synchronization Barrier* (`ESB`) instruction.
- An implicit ESB operation that is inserted after exception entry, exception return, and lazy stacking. This feature is enabled by setting AIRCR.IESB. For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.
- Two ID registers, ERRDEVID and ID_PFR0. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.
- A fault status register, RFSR, that is dedicated to RAS events. For more information on:
  - RAS events, see 11.1.1 Cortex-M85 RAS events on page 198.
  - RFSR, see 11.4.7 RFSR, RAS Fault Status Register on page 216.
- A summary register indicating the nodes that have detected RAS events, ERRGSR. For more information on this register, see 11.4.5 ERRGSR0, RAS Fault Group Status Register on page 214. A node is a unit that can detect RAS events, and for Cortex®-M85, a node is the entire

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

processor. Therefore, all RAS events are logged in the same location and the processor supports a single error record.

- Each node has one set of Error Record Registers that can store information about the last RAS event that the node has detected.
  The RAS Error Record Registers are independent of the Error Bank Registers, although they have some common behavior. Either or both of the register types can be used by system software that is handling errors. However, for compatibility across other devices and systems that implement the RAS Extension, the RAS programmers' model must be considered. The RAS Error Record Registers are described in 11.4 RAS Extension registers on page 207 and the Error Bank Registers are described in 5.12 Error bank registers on page 78.

Tip

For a complete description of RAS error types and the information on RAS errors that are produced at the node, see the *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

11.1.1 Cortex®-M85 RAS events

The *Reliability, Availability, and Serviceability* (RAS) Extension provides a standard model for recording and reporting errors which might occur during the operation of a system.

In the Cortex®-M85 processor, the following are considered as RAS events:

- L1 instruction cache *Error Correcting Code* (ECC) errors.
- L1 data cache ECC errors.
- TCM ECC errors.

Note

For more information on how these RAS events are detected and handled in the Cortex®-M85 processor, see 11.2 ECC memory protection behavior on page 198 to get an overview on how instruction cache, data cache, and TCM ECC errors are handled.

11.2 ECC memory protection behavior

Error Correcting Code (ECC) memory protection is optional. At implementation, you can configure the Cortex®-M85 processor to include ECC or not using the Verilog parameter, `ECC`. At Cold reset, if the Cortex®-M85 processor is configured with ECC, you can control whether ECC is enabled or not using the static configuration signal **INITECCEN**. **INITECCEN** must only be changed when the processor is powered down and in Cold or Warm reset.

ECC memory protection includes the following protection features:

- Data protection

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Address decoder protection
- White noise protection, which involves protection against faults in the RAM that might also result in no entry being selected and therefore, resulting in reading either all zeros or all ones.

11.2.1 ECC schemes and error type terminology

The Cortex®-M85 processor supports two *Error Correcting Code* (ECC) schemes to detect errors.

ECC schemes

SECDED

Single Error Correct Double Error Detect (SECDED) is used on the L1 data cache and TCM RAMs. The SECDED scheme also provides information on how to correct the error.

DED

Double Error Detect (DED) is used on the L1 instruction cache RAMs. The DED scheme detects single bit and double bit errors. The instruction cache does not need a correction mechanism or scheme because the contents must always be consistent with external memory. Therefore, the processor automatically invalidates the instruction cache RAM to correct its contents.

In the Cortex®-M85 processor, the ECC schemes can also support detection of some multi-bit errors where more than two bits are incorrect. Where possible, RAM location information is included in the ECC code to allow fault detection in the RAM address decoder logic.

Error type terminology

The following error type terminology is used in this manual in the context of ECC:

Single-bit error

An error where only one bit of the data or ECC code is incorrect. These errors can usually be corrected.

Note

ECC errors detected in the address field are treated as multi-bit errors, because this indicates that an incorrect location has been read and all of the data is wrong.

Multi-bit error

An error in which any one of the following is true:

- More than one bit of data or ECC code is incorrect.
- An error is detected in one or more address bits.
- The RAM read value is all ones or all zeros.

Corrected error (CE)

An ECC error that is detected by hardware and that hardware can correct. These are:

- Single bit errors, which can be corrected inline by flipping the faulty bit.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- All errors which can be corrected by refetching the data from external memory. This includes all instruction cache errors and all data cache errors when the cache line can be guaranteed to be clean.

For more information on Corrected errors (CEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

Uncorrected error (UE)

An ECC error that cannot be corrected or deferred. These are multi-bit errors:

- From the TCMs.
- In an L1 dirty data cache data RAM where it is not guaranteed that the cache line is clean. This includes the case where the ECC indicates that the RAM location is incorrect.
- In an L1 dirty data cache tag RAM where it is not guaranteed that the cache is clean. This includes the case where the ECC indicates that the RAM location is incorrect.

For more information on Uncorrected errors (UEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

11.2.2 Enabling ECC

If configured in the processor, *Error Correcting Code* (ECC) is enabled at reset using the input signal **INITECCEN**.

For more signal information, see C.29 Error interface signals on page 372. For more information on MSCR, see 5.13 MSCR, Memory System Control Register on page 84.

If ECC is enabled out of reset, the L1 cache must be invalidated before it is enabled to avoid spurious ECC errors being detected because of a mismatch between the data and ECC in the RAM. Automatic instruction and data cache invalidation can be enabled at reset by tying the input signal **INITL1RSTDIS** LOW. For more signal information, see C.29 Error interface signals on page 372. For more information on automatic cache invalidation, see 10.9.4 Automatic cache invalidation at reset on page 188.

Note

Software can determine whether ECC is configured and enabled by reading MSCR.ECCEN. However, software cannot enable ECC.

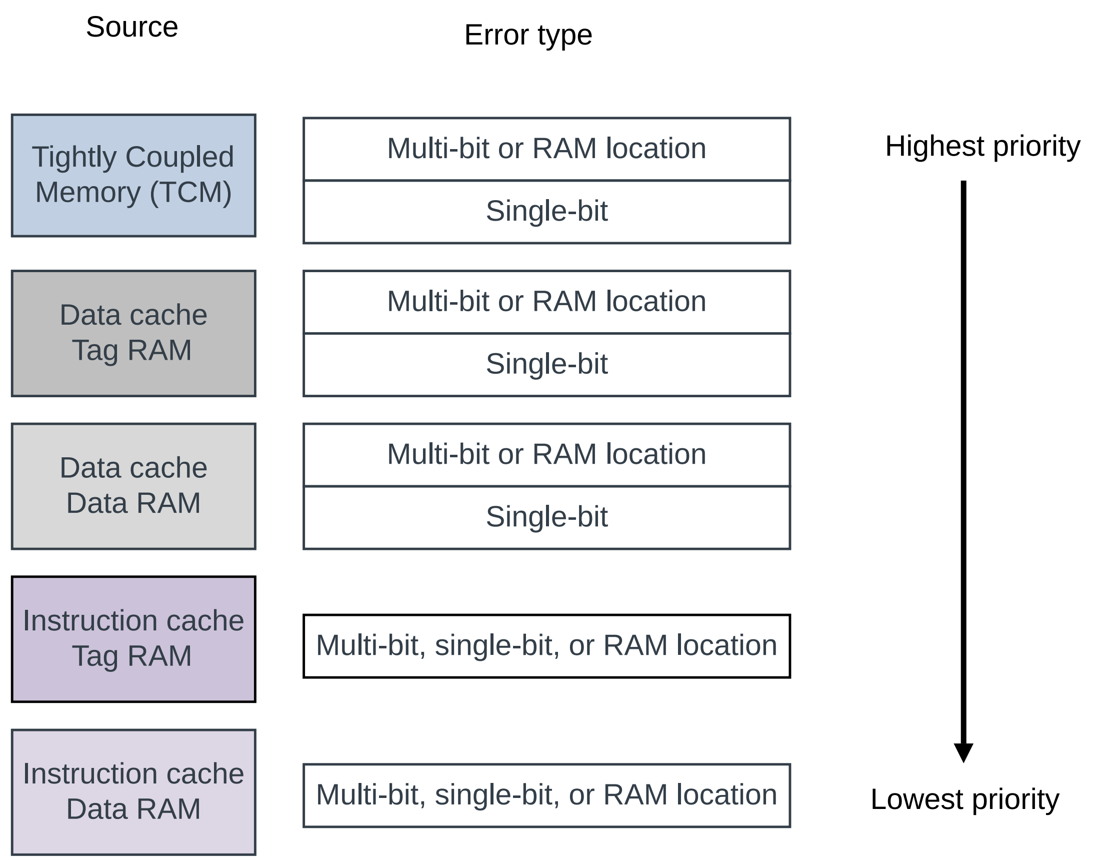
11.2.3 Error detection and processing

The Cortex®-M85 processor core is responsible for error detection and processing. Multiple errors can occur simultaneously, therefore, the processor prioritizes the error processing based on the source.

The following figure shows the prioritization of error processing that occurs in the order of decreasing priority.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 11-1: Error processing prioritization



The errors in the *Data Tightly Coupled Memory* (DTCM) always have higher priority than the errors in the *Instruction Tightly Coupled Memory* (ITCM).

11.2.3.1 Error processing in the L1 data and instruction cache

The cache tag and data RAMs are read during various operations that the Cortex®-M85 processor carries out.

The following table lists these operations.

Table 11-1: L1 cache RAM access classes

Access type	RAM block read	Notes
Instruction fetch	Instruction tag and data RAM	Two tag banks and up to two data banks

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Access type	RAM block read	Notes
Load request	Data tag and data RAM	4 tag banks and up to four data banks
Dirty line eviction	Data RAM	Entire line is read in parallel
Store buffer address read	Data tag RAM	Four tag banks
Store buffer data read	Data data RAM	Only used for *Read-Modify-Write* (RMW). RMW is used when the processor writes a partial word when ECC is enabled. Store operations to a cache line, which are less that 64 bits of data must read the data RAM to construct the ECC to write back. This is based on the combination of the current and new data. This read operation can result in an error being detected in the data RAM.
Data cache maintenance	Tag RAM and data RAM	Tag RAM read for address-based and clean operations. Data RAM read for clean evictions.

The error processing operations are:

Instruction fetch

All *Error Correcting Code* (ECC) errors on instruction fetches are processed by invalidating the tag RAM and refetching the line from external memory.

Corrected errors in the L1 data cache for load and store operations

Corrected errors (CE) in the L1 data cache that are detected on load, store, and cache maintenance operations are processed by cleaning (if required) and invalidating the location.

Load operations will stall on an ECC error until the error is processed. Thereafter, a linefill request for the data is made, returning the data to the load as it is also re-allocated into the cache.

Store operations to Write-Allocate memory request a linefill after the error has been processed and then merge the write data into the line as it is allocated to the cache. Store operations to a line in the cache which write less than 64 bits of data must read the data RAM to construct the ECC to write-back, based on a combination of the current and new data. This read operation can result in an error being detected in the data RAM.

Cache maintenance operations

Data cache maintenance operations which operate on an address read all four tag RAMs to check for a match. Instruction cache maintenance operations which operate on an address read two tag RAMs to check for a match. Therefore, they can potentially detect multiple errors unrelated to the requested location. The operation automatically cleans and invalidates all detected errors in sequence. Cache maintenance invalidate by set/way location carried out by Non-secure code always reads the tag because it might contain a dirty line associated with a Secure address, and therefore, it must be cleaned to prevent data loss before being invalidated. The behavior of cache maintenance operations in Non-secure state is described in 10.9.3 Cache maintenance operations on page 186.

Dirty line eviction

In all cases where a line is evicted, the data RAM associated with the entire line is read out of the cache. Any error detected in this read is corrected inline before being written back to the external memory through the *Master AXI* (M-AXI) interface. If a multi-bit error

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

is detected in the data, the line is marked as poisoned and an imprecise BusFault is raised if MSCR.EVECCFAULT is set.

Multiple errors are processed according to the priority listed in 11.2.3 Error detection and processing on page 200.

If data is lost because of a multi-bit ECC error, then an Imprecise BusFault is generated under the following conditions:

- If a data cache eviction is performed, and a multi-bit error is detected in the data RAM and MSCR.EVECCFAULT is set.
- If a data cache line is invalidated because of a multi-bit error detected in the Tag RAM, and MSCR.DCCLEAN is not set.

Although loads do not directly cause BusFaults, they cause ECC maintenance behavior that triggers a BusFault if data is lost. Additionally, if any load sees an ECC error the pipe is flushed, and the load cannot progress until the ECC maintenance has finished. This guarantees that the core does not consume erroneous data until an Imprecise BusFault has been generated.

Although loads do not directly cause BusFaults, they cause ECC maintenance behavior that triggers a BusFault if data is lost. Additionally, if any load sees an ECC error the pipe is stalled, and the load cannot progress until the ECC maintenance has finished. This guarantees that the core does not consume erroneous data until an Imprecise BusFault has been generated.

A multi-bit error on the data cache tag when MSCR.DCCLEAN is asserted is always correctable as the corresponding cache line cannot contain any dirty data.

A multi-bit error on the data cache data when MSCR.EVECCFAULT is deasserted is considered Deferred (DE), because when that line is evicted, it is marked as poisoned. MSCR.EVECCFAULT being deasserted implies that the system supports poisoning.

Any other case of multi-bit errors in the data cache is considered Uncorrected.

11.2.3.2 Error processing in the TCMs

Error detection and correction are carried out on each of the individual TCMS, that is, ITCM, D0TCM, D1TCM, D2TCM, and D3TCM. Accesses to each of the interfaces are treated in the following way:

- Correctable errors detected during instruction fetch result in the read being repeated by refetching the instruction address. Load operations that encounter correctable errors stall the pipeline and wait for the corrected data. The corrected data is written back to the TCM.
- Correctable errors from read requests on the *Slave AHB* (S-AHB) are corrected inline and returned to the system on completion of the transaction.
- Write requests to the TCM with an access size smaller than a complete TCM ECC code granule (doubleword for ITCM, word for DTCM) or with non-contiguous bytes from S-AHB or *M-profile Vector Extension* (MVE) operations must carry out a *Read-Modify-Write* (RMW) sequence to the TCM. Correctable errors detected during the sequence are corrected inline before the complete

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

store word is written back to the TCM. Uncorrectable errors that are detected on the read phase of an RMW sequence cause the write phase to be abandoned, and the address is marked as poisoned in the error bank register. If the location is read again, a precise BusFault is raised.

- When ECC is enabled, an instruction fetch or load operations might raise a precise BusFault exception, if an *Uncorrected error* (UE) is detected.

Note

When ECC is enabled, before performing a byte, halfword, or unaligned word write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned doubleword write to the location. Arm recommends that all TCM locations are initialized in this manner by boot code.

11.2.4 Error reporting

Error reporting is done using both registers and output signals.

Corrected errors

Corrected errors (CE) are always transparent to program flow. For more information on Corrected errors (CEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

Uncorrected errors

Uncorrected errors (UEs) can result in a precise or imprecise BusFault. If an exception occurs, the source of the error can be determined using the AFSR and RFSR.

An imprecise BusFault is raised when a UE is found in the data cache data RAM during an eviction. If the system supports poisoning, clearing MSCR.EVECCFAULT disables this error. An imprecise BusFault is also raised when a UE is found in the data cache tag RAM and MCSR.DCCLEAN is not set and this type of BusFault cannot be disabled. For more information on Uncorrected errors (UEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

Errors detected on accesses to the TCMs never result in an imprecise BusFault.

Errors on the L1 instruction cache, L1 data cache, and TCMs

Errors detected in the L1 instruction cache, L1 data cache, and TCMs are reported on the following external error interface output signals:

- **DMEV0**
- **DMEV1**
- **DMEV2**
- **DMEL0[2:0]**
- **DMEL1[2:0]**
- **DMEI0[25:0]**
- **DMEI1[25:0]**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Up to two errors can be reported on the same cycle. If multiple simultaneous errors occur, the priority scheme for reporting is followed. The reporting priority is described in 11.2.3 Error detection and processing on page 200. If up to two errors occur, the location and error class is indicated in **DMELn** and **DMEIn** respectively, and **DMEVn** is asserted. If more than two errors occur, then only information about the two highest priority errors are reported and **DMEV2** is asserted to indicate further information is not available.

For more signal information, see C.29 Error interface signals on page 372.

Note

A particular ECC error might be reported multiple times on the DME bus.

Error bank registers

The processor includes internal error bank registers which do the following:

- Record the two most recent errors detected.
- Isolate the system from hard errors in the RAM which cannot be corrected by invalidating or overwriting with correct data.

Two error bank registers are included for each source of errors:

- IEBR0 and IEBR1 for the L1 instruction cache.
- DEBR0 and DEBR1 for the L1 data cache.
- TEBR0, TEBR1, TEBRDATA0, and TEBRDATA1 that are shared across the ITCM and DTCM.

Error bank behavior

When an error bank contains a valid entry, any errors detected from the associated RAM address are ignored.

L1 instruction and data cache

For the L1 instruction and data cache, the RAM addresses are masked on a cache lookup and no longer used for allocating a line on a miss, isolating the processor from any potential hard errors in the RAM which could cause incorrect behavior even if corrected data is written from external memory.

TCMs

For TCMs, each TCM error bank contains a 64-bit data register TEBRDATAn – 64 bits are used for ITCM and only low 32 bits are used for DTCM. When a single-bit TCM fault is detected and the error bank is allocated, the corrected data is written to the data register and the TCM memory. Any subsequent read returns the result directly from TEBRDATAn. Writes to an address associated with a valid TCM Error bank is written to both the TEBRDATAn and the TCM RAM to maintain consistency if the error bank is reallocated or cleared by software. If a multi-bit error is detected on a read from the TCM RAM, the error bank TEBRn.POISON field is set. When this field has been set any subsequent read requests to the TCM which matches the error bank address, it will result in an error. A precise BusFault will be raised for a load request from the processor and **HRESP** is asserted on a read on the *Slave AHB* (S-AHB) interface.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Write accesses from store instructions or S-AHB to TCM that match an error bank register with TEBRn.POISON set do not raise a fault. DTCM clears the TEBRn.POISON field with an aligned 32-bit write and ITCM clears TEBRn.POISON with an aligned 64-bit write to the address associated with the TCM error bank register. The behavior of the poison feature in the TCM error bank register allows hard multi-bit errors to be patched by software. For example:

1. Load from the TCM at an address detects a multi-bit *Error Correcting Code* (ECC) error. TEBRn is allocated, TEBRn.POISON is set, and a fault is raised.
2. Patch write data of 32 bits is stored to the TCM at that address – 64 bits are used for ITCM and only low 32 bits are used for DTCM. TEBRDATAn and TCM memory are updated and TEBRn.POISON is cleared.
3. Subsequent read and write transactions to that address are completed as expected.

If this sequence is applied, the failing TCM RAM entry is isolated and normal execution can continue when the write is applied, even when the error is Hard and so cannot be cleared by a patch directly to the RAM. Between steps 1 and 2, read and write transactions with a size less than 32 bits for DTCM or less than 64 bits for ITCM continue to raise a fault because the address has not been patched.

The error bank registers are updated when an ECC error from the associated RAM controller has been processed and remains valid until either a subsequent error is detected and processed, or a direct software write to the bank is carried out to clear the data.

Invalid error banks are always allocated in preference to valid error banks. If both error banks contain valid data new errors are allocated using a round-robin approach. Error banks can be locked from being overwritten by writing to the LOCKED field in the error bank register.

Note

Locking both error banks is prevented by the hardware to avoid potential deadlocks.

The error bank registers are only cleared on Cold reset and retain their content on system reset.

11.2.5 Address decoder protection and white noise protection

The Cortex®-M85 processor includes address decoder protection and white noise protection.

Address decoder protection

Address decoder protection detects some of the errors that might occur because of a failure in the address decoder in a RAM instance. A fault in a RAM address decoder circuit might result in the wrong RAM entry being selected, which typically contains data and ECC that are self-consistent. Therefore, an ECC error on the data is not generated in this case, but the wrong data is read from the RAM.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

White noise protection

A fault in a RAM might result in no entry being selected, which might result in reading either all zeros or all ones. Protection against such faults is white noise protection.

11.3 RAS memory barriers

The *Reliability, Availability, and Serviceability* (RAS) extension supports the *Error Synchronization Barrier* (`ESB`) instruction.

When this instruction is executed, all outstanding errors which have been detected but not reported are visible to the software running on the system. In the Cortex®-M85 processor, this instruction behaves in the same way as the *Data Synchronization Barrier* (`DSB`) instruction. When executed, all outstanding requests in the memory system are completed before the `ESB` instruction completes and any required BusFault exceptions are raised.

The RAS architecture supports another *Error Synchronization Barrier* (ESB) operation, which is implicit, that is, the *Implicit Error Synchronization Barrier* (IESB) operation. This feature is enabled by setting the AIRCR.IESB bit. When enabled, a barrier is inserted after the end of any register stacking or unstacking sequence associated with exception entry, exit, or floating-point register lazy stacking. Execution is halted in the processor until all outstanding transactions, including the stacking sequence have completed and any errors have been reported. The implicit barrier allows software to isolate an error during context switches, with RAS events always being reported in the old context.

Caution

Use IESB carefully because waiting for outstanding transactions to complete on exception entry can increase interrupt latency, particularly if an AXI access associated with the interrupted context takes many cycles to complete. The feature is disabled by default, with AIRCR.IESB set to 0 out of reset.

For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.

11.4 RAS Extension registers

The Cortex®-M85 processor implements the *Reliability, Availability, and Serviceability* (RAS) features to ensure correct operation in environments where functional safety and high-availability are critical. The RAS features can be controlled using the RAS Extension registers.

The following table lists the RAS Extension registers.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 11-2: RAS Extension registers

Address	Name	Type	Reset value	Description
0xE0005000	ERRFR0	RO	0x00000101 **Note:** 0x00000000, if the processor is not configured with *Error Correcting Code* (ECC).	11.4.1 ERRFR0, RAS Error Record Feature Register on page 208
0xE0005008	ERRCTRL0	-	-	This register is **RES0**.
0xE0005010	ERRSTATUS0	RW	**UNKNOWN**	11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register on page 209
0xE0005018	ERRADDR0	RO	**UNKNOWN**	11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers on page 212
0xE000501C	ERRADDR20	RO	**UNKNOWN**	11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers on page 212
0xE0005020	ERRMISC00	-	-	This register is **RES0**.
0xE0005024	ERRMISC10	RO	**UNKNOWN**	11.4.4 ERRMISC10, Error Record Miscellaneous Register 10 on page 213
0xE0005028	ERRMISC20	-	-	This register is **RES0**.
0xE000502C	ERRMISC30	-	-	This register is **RES0**.
0xE0005030	ERRMISC40	-	-	This register is **RES0**.
0xE0005034	ERRMISC50	-	-	This register is **RES0**.
0xE0005038	ERRMISC60	-	-	This register is **RES0**.
0xE000503C	ERRMISC70	-	-	This register is **RES0**.
0xE0005E00	ERRGSR0	RO	0x00000000	11.4.5 ERRGSR0, RAS Fault Group Status Register on page 214
0xE0005FC8	ERRDEVID	RO	0x00000001 **Note:** 0x00000000, if the processor is not configured with ECC.	11.4.6 ERRDEVID, RAS Error Record Device ID Register on page 215
0xE000EF04	RFSR	RW	**UNKNOWN**	11.4.7 RFSR, RAS Fault Status Register on page 216

11.4.1 ERRFR0, RAS Error Record Feature Register

The *Reliability, Availability, and Serviceability* (RAS) ERRFR0 register describes the RAS features that are supported.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

Configurations

This register is always implemented.

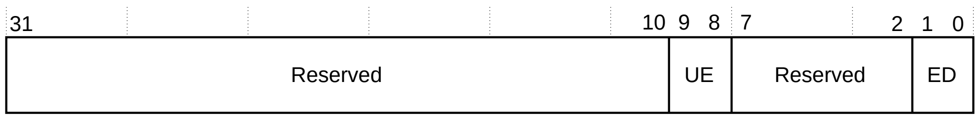
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRFR0 bit assignments.

Figure 11-2: ERRFR0 bit assignments



The following table describes the ERRFR0 bit assignments.

Table 11-3: ERRFR0 bit assignments

Field	Name	Type	Description
[31:10]	Reserved	-	**RES0**
[9:8]	UE	RO	Enable Uncorrected error (UE) reporting as an external abort. `0b01` External abort response for uncorrected errors enabled. This field indicates that uncorrectable errors cause BusFault exceptions.
[7:2]	Reserved	-	**RES0**
[1:0]	ED	RO	Error reporting and logging. `0b01` Reporting and logging always enabled. This field indicates that logging and reporting of errors cannot be disabled.

11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register

The Arm®v8.1-M *Reliability, Availability, and Serviceability* (RAS) ERRSTATUS0 register contains information about the *Reliability, Availability, and Serviceability* (RAS) event that is currently logged in record 0.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

Configurations

This register is always implemented.

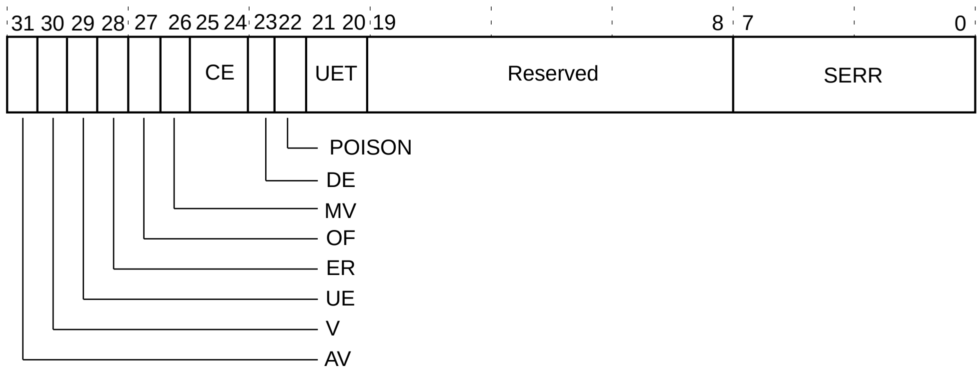
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

The register is not banked between Security states. The read/write behavior depends on the individual fields. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRSTATUS0 bit assignments.

Figure 11-3: ERRSTATUS0 bit assignments



The following table describes the ERRSTATUS0 bit assignments.

Table 11-4: ERRSTATUS0 bit assignments

Field	Name	Type	Description
[31]	AV	RW	Address valid. **0b0** ERRADDR0 is not valid. **0b1** ERRADDR0 is valid. ERRADDR0 is valid only if: • A precise BusFault caused the RAS event. • A TCM *Error Correcting Code* (ECC) error caused the RAS event. This bit is write-one-to-clear.
[30]	V	RW	Status valid. **0b0** ERRSTATUS0 is not valid. **0b1** ERRSTATUS0 is valid. This field is set to 1 on any RAS event. This bit is write-one-to-clear.
[29]	UE	RW	Uncorrected errors (UEs). **0b0** No uncorrectable errors detected. **0b1** At least one uncorrectable error is detected. This bit is write-one-to-clear.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[28]	ER	RW	Error reported. `0b0` No BusFault caused by RAS event has occurred. `0b1` BusFault caused by RAS event has occurred. This bit is write-one-to-clear.
[27]	OF	RW	Overflow. `0b0` At most one RAS event has occurred since the last time ERRSTATUS0.V was cleared. `0b1` At least two RAS events have occurred since the last time ERRSTATUS.V was cleared. These events might have occurred at the same time. This bit is write-one-to-clear.
[26]	MV	RW	Miscellaneous registers valid. `0b0` ERRMISC0 is not valid. `0b1` ERRMISC0 is valid. This field is set to 1 on any RAS event. This bit is write-one-to-clear.
[25:24]	CE	RW	Corrected errors. `0b00` Corrected errors (CEs) have not been detected. `0b10` At least one Corrected error (CE) has been detected. This bit is write-one-to-clear.
[23]	DE	RW	Deferred errors. `0b0` No errors were deferred. `0b1` At least one error was deferred. This bit is write-one-to-clear.
[22]	POISON	RW	`0b0` No BusFault due to RPOISON or TEBRn.POISON has occurred. `0b1` At least one BusFault due to RPOISON or TEBRn.POISON has occurred.
[21:20]	UET	RW	Uncorrectable error type. `0b00` Uncorrectable error, Uncontainable error (UC). This is for any uncorrectable error that caused an asynchronous BusFault `0b11` Uncorrectable error, Recoverable error (UER). This is for an uncorrectable error that caused a synchronous BusFault These bits are write-one-to-clear (`0b11`)
[19:8]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[7:0]	SERR	RW	Architecturally-defined primary error code. **0** No error. **2** TCM ECC error. **6** L1 data cache or instruction cache data RAM ECC error. **7** L1 data cache or instruction cache tag RAM ECC error. **21** Poison BusFault due to RPOISON or TEBRn.POISON. The Cortex®-M85 processor does not use the other values of this field.

11.4.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers

The *Reliability, Availability, and Serviceability* (RAS) ERRADDR0 and ERRADDR20 registers contain information about the address of the *Reliability, Availability, and Serviceability* (RAS) event in record 0.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

This register ignores writes if ERRSTATUS0.AV is set to 1.

Configurations

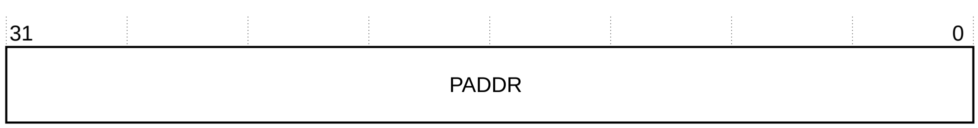
These registers are always implemented.

Attributes

These registers are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRADDR0 bit assignments.

Figure 11-4: ERRADDR0 bit assignments



The following table describes the ERRADDR0 bit assignments.

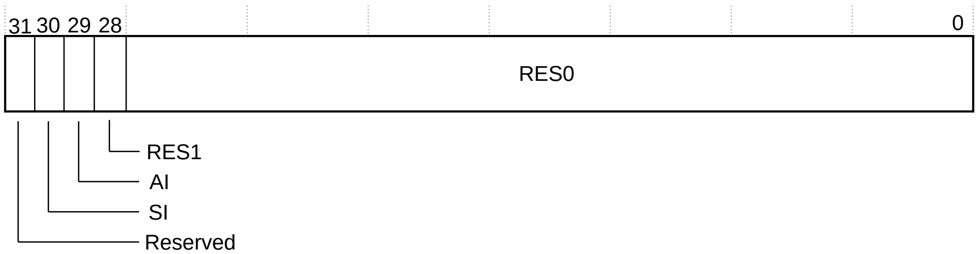
Table 11-5: ERRADDR0 bit assignments

Field	Name	Type	Description
[31:0]	PADDR	RW	Address of the RAS event. This is the address associated with the memory access that observed *Error Correcting Code* (ECC) error. This field is not valid if ERRADDR20.AI is `0b1`.

The following figure shows the ERRADDR20 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 11-5: ERRADDR20 bit assignments



The following table describes the ERRADDR20 bit assignments.

Table 11-6: ERRADDR20 bit assignments

Field	Name	Type	Description
[31]	Reserved	-	**RES0**
[30]	SI	RO	Security information incorrect. **`0b1`** NS bit is not valid. The security information is never guaranteed to be correct.
[29]	AI	RO	Address incorrect. **`0b0`** PADDR is valid. **`0b1`** PADDR is not valid. PADDR is valid only if: • The RAS event was a precise BusFault. • The RAS event was associated with a TCM ECC error. **Note:** If software clears ERRSTATUS.AV, then ERRADDR20.AI is set to `0b1` to invalidate the address.
[28]	Reserved	-	**RES1**
[27:0]	Reserved	-	**RES0**

11.4.4 ERRMISC10, Error Record Miscellaneous Register 10

The ERRMISC10 register is an **IMPLEMENTATION DEFINED** error syndrome register for the event in record 0.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

Configurations

This register is always implemented.

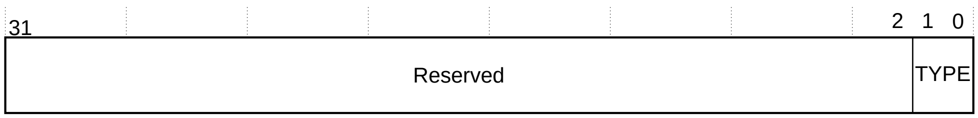
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRMISC10 bit assignments.

Figure 11-6: ERRMISC10 bit assignments



The following table describes the ERRMISC10 bit assignments.

Table 11-7: ERRMISC10 bit assignments

Field	Name	Type	Description
[31:2]	Reserved	-	**RES0**
[1:0]	TYPE	RO	Indicates the type of *Reliability, Availability, and Serviceability* (RAS) event logged. `0b00` L1 instruction cache ECC. `0b01` L1 data cache ECC. `0b10` TCM ECC found by load or store executed by the processor. `0b11` TCM ECC found by access from *Slave AHB* (S-AHB).

Note

In the Cortex®-M85 processor, only ERRMISC10 is implemented. ERRMISC00 and ERRMISC20-ERRMISC70 are **RES0**.

11.4.5 ERRGSR0, RAS Fault Group Status Register

The ERRGSR0 register summarizes the valid error records. The Cortex®-M85 processor only supports one error record, therefore, only one bit of ERRGSR is active.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

Configurations

This register is always implemented.

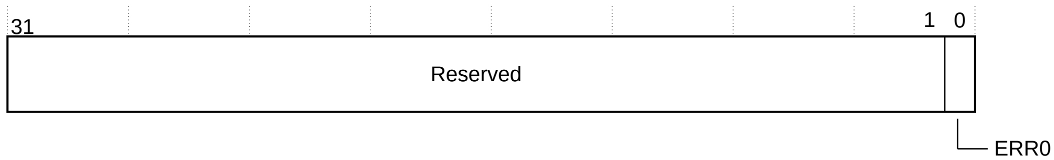
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRGSR0 bit assignments.

Figure 11-7: ERRGSR0 bit assignments



The following table describes the ERRGSR0 bit assignments.

Table 11-8: ERRGSR0 bit assignments

Field	Name	Type	Description
[31:1]	Reserved	-	RES0
[0]	ERR0	RO	Error record 0 is valid.

11.4.6 ERRDEVID, RAS Error Record Device ID Register

The *Reliability, Availability, and Serviceability* (RAS) ERRDEVID register contains the number of error records that an implementation supports. The Cortex®-M85 processor supports a single error record with index 0 if *Error Correcting Code* (ECC) is configured or there are no error records.

Usage constraints

Unprivileged access results in a BusFault exception.

This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL_S.UIDAPEN or DAUTHCTRL_NS.UIDAPEN is set.

Configurations

This register is always implemented.

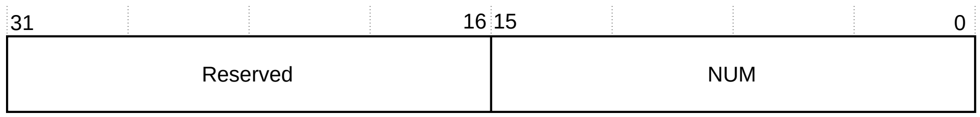
Attributes

This register is not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the ERRDEVID bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 11-8: ERRDEVID bit assignments



The following table describes the ERRDEVID bit assignments.

Table 11-9: ERRDEVID bit assignments

Field	Name	Type	Description
[31:16]	Reserved	-	**RES0**
[15:0]	NUM	RO	Maximum Error Record Index+1 `0x0001` If ECC is configured, then one error record with index 0. `0x0000` If ECC is not configured, then there are no error record registers. **Note:** • ECC is configured using the Verilog parameter `ECC` and enabled by driving the input signal INITECCEN to 1. • ERRDEVID[0] always reads the same value as MSCR.ECCEN.

11.4.7 RFSR, RAS Fault Status Register

The RFSR reports the fault status of *Reliability, Availability, and Serviceability* (RAS) related faults from *Error Correcting Code* (ECC) errors that are detected in the L1 instruction cache, data cache, and TCM.

Usage constraints

If AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state.

If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

Configurations

This register is always implemented.

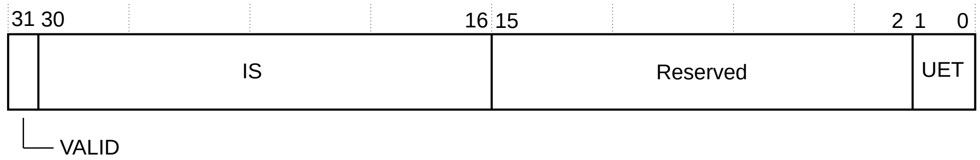
Attributes

This is register are not banked between Security states. See 5.10 IMPLEMENTATION DEFINED registers summary on page 68 for more information.

The following figure shows the RFSR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 11-9: RFSR bit assignments



The following table describes the RFSR bit assignments.

Table 11-10: RFSR bit assignments

Bit	Name	Type	Description
[31]	Valid	RW	Indicates whether the register is valid. This bit is write-one-to-clear and therefore, it is cleared by writing 1. Writes of zero are ignored.
[30:16]	IS	RW	IMPLEMENTATION-DEFINED syndrome. Indicates the type of RAS exception that has occurred. `0x0` L1 instruction cache ECC. `0x1` L1 data cache ECC. `0x2` TCM ECC.
[15:2]	Reserved	-	RES0.
[1:0]	UET	RW	Error type. `0b00` Uncontainable error (UC). RAS exception is imprecise. `0b11` Recoverable error (UER). RAS exception is precise. For more information on error types, see the 11.2.1 ECC schemes and error type terminology on page 199.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

12 Nested Vectored Interrupt Controller

This chapter describes the *Nested Vectored Interrupt Controller* (NVIC).

12.1 NVIC features

The Cortex®-M85 processor *Nested Vectored Interrupt Controller* (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

The NVIC is responsible for:

- Maintaining the current execution priority of the Cortex®-M85 processor.
- Maintaining the pending and active status of all exceptions that are supported.
- Invoking preemption when a pending exception has priority.
- Providing wakeup signals to wakeup the Cortex®-M85 processor from deep sleep mode.
- Providing support to the *Internal Wakeup Interrupt Controller* (IWIC) and *External Wakeup Interrupt Controller* (EWIC).
- Providing priority and exception information to other processor components.

The NVIC in the Cortex®-M85 processor allows up to 496 exceptions, of which, 480 can be regular external interrupts.

12.2 Registers associated with interrupt control and behavior

Registers associated with interrupt control and interrupt behavior are found in the following categories.

Table 12-1: Interrupt control and behavior registers

Register summary	Registers	Description
System control block	• ICSR • AIRCR • SHPR1-3	5.1 System control register summary on page 49
Implementation control block	ICTR	5.7 Implementation control register summary on page 64
Software Interrupt Generation	STIR	12.4 Software Interrupt Generation register summary on page 219
SysTick Timer	• SYST_CSR • SYST_RVR • SYST_CVR • SYST_CALIB	12.5 SysTick Timer register summary on page 220

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

12.3 NVIC register summary

The *Nested Vectored Interrupt Controller* (NVIC) registers can be accesses through the *Internal Private Peripheral Bus* (IPPB) interface. Each of the NVIC registers is 32 bits wide.

The NVIC_ISERn, NVIC_ICERn, NVIC_ISPRn, NVIC_ICPRn, NVIC_IABRn, and NVIC_IPRn registers are not banked between Security states. If an interrupt is configured as Secure in the NVIC_ITNSn register, any access to the corresponding NVIC_ISERn, NVIC_ICERn, NVIC_ISPRn, NVIC_ICPRn, NVIC_IABRn, or NVIC_IPRn registers from Non-secure are treated as RAZ/WI.

For more information on the NVIC registers listed in the following table, see *Arm®v8-M Architecture Reference Manual*.

Table 12-2: NVIC register summary

Address offset	Name	Type	Reset value	Description
`0xE000E100-0xE000E13C`	NVIC_ISER0-NVIC_ISER15	RW	`0x00000000`	Interrupt Set-Enable Registers
`0xE000E180-0xE000E1BC`	NVIC_ICER0-NVIC_ICER15	RW	`0x00000000`	Interrupt Clear-Enable Registers
`0xE000E200-0xE000E23C`	NVIC_ISPR0-NVIC_ISPR15	RW	`0x00000000`	Interrupt Set-Pending Registers
`0xE000E280-0xE000E2BC`	NVIC_ICPR0-NVIC_ICPR15	RW	`0x00000000`	Interrupt Clear-Pending Registers
`0xE000E300-0xE000E33C`	NVIC_IABR0-NVIC_IABR15	RO	`0x00000000`	Interrupt Active Bit Register
`0xE000E380-0xE000E3BC`	NVIC_ITNS0-NVIC_ITNS15	RW	`0x00000000`	Interrupt Target Non-secure Registers **Note:** These registers are Secure only. They are RAZ/WI when accessed from Non-secure state.
`0xE000E400-0xE000E5DC`	NVIC_IPR0-NVIC_IPR119	RW	`0x00000000`	Interrupt Priority Registers

12.4 Software Interrupt Generation register summary

The following table shows the architecturally defined Software Interrupt Generation register.

Table 12-3: Software Interrupt Generation register summary

Address offset	Name	Type	Reset value	Description
`0xE000EF00`	STIR	WO	`0x00000000`	Software Triggered Interrupt Register. For more information, see *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

12.5 SysTick Timer register summary

The following table shows the architecturally defined SysTick Timer registers.

Note

For more information on the architectural registers listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

Table 12-4: SysTick Timer register summary

Address offset	Name	Type	Reset value	Description
`0xE000E010`	SYST_CSR	RW	`0x00000000`	SysTick Control and Status Register
`0xE000E014`	SYST_RVR	RW	`0x00000000`	SysTick Reload Value Register
`0xE000E018`	SYST_CVR	RW	`0x00000000`	SysTick Current Value Register
`0xE000E01C`	SYST_CALIB	RO	`0x00000000`	SysTick Calibration Value Register

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

13 External coprocessors

This chapter describes the interface and programmer's model for connecting and using external coprocessors.

13.1 External coprocessors features

The Cortex®-M85 processor supports an external coprocessor interface which allows the integration of tightly coupled accelerator hardware with the processor. The programmers model allows software to communicate with the hardware by using architectural coprocessor instructions.

The external coprocessor interface:

- Supports low-latency data transfer from the processor to and from the accelerator components.
- Provides a mechanism for you to extend the capabilities of the Cortex®-M85 processor.
- Supports up to eight separate coprocessors, CP0-CP7, depending on your implementation. The remaining coprocessor numbers, CP8-CP15, are reserved. CP10 and CP11 are always reserved for floating-point or *M-profile Vector Extension* (MVE) functionality. For more information, see the *Arm®v8-M Architecture Reference Manual*. The Cortex®-M85 processor system can configure which coprocessor is included in Secure and Non-secure states.

13.2 Operation

The external coprocessor interface provides control and data channels for up to eight separate coprocessors. The external devices are provided with information about privilege and Security state of the processor with the instruction type and associate register and operation fields that the architecture defines. The following instruction types are supported:

- Register transfer from the Cortex®-M85 processor to the coprocessor `MCR`, `MCRR`, `MCR2`, `MCRR2`.
- Register transfer from the coprocessor to the Cortex®-M85 processor `MRC`, `MRRC`, `MRC2`, `MRRC2`.
- Data processing instructions `CDP`, `CDP2`.

The interface provides a handshake mechanism to indicate to the coprocessor that an instruction has been committed in the processor and can no longer be interrupted. Additionally, it can stall the processor in a way that it can always be interrupted (BUSYWAIT) and to indicate that an error has occurred while waiting for an UNDEFINSTR UsageFault.

Note

- The regular and extension forms of the coprocessor instructions for example, `MCR` and `MCRR2`, have the same functionality but different encodings. The two encoding values differ by a single bit, bit [12]. For more information, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The `MRC` and `MRC2` instructions support the transfer of APSR.NZVC flags when the processor register field is set to PC, for example Rt == `0xF`.

13.3 Data transfer rates

The following table lists the ideal data transfer rates for the coprocessor interface. This means that the coprocessor responds to an instruction immediately and does not BUSYWAIT. The ideal data transfer rates are sustainable if the corresponding coprocessor instructions are executed consecutively.

Table 13-1: Ideal data transfer rates for the coprocessor interface

Instructions	Direction	Ideal data rate
`MCR`, `MCR2`	Processor to coprocessor	32 bits per cycle
`MRC`, `MRC2`	Coprocessor to processor	32 bits per cycle
`MCRR`, `MCRR2`	Processor to coprocessor	64 bits per cycle
`MRRC`, `MRRC2`	Coprocessor to processor	64 bits per cycle

13.4 Coprocessor instruction restrictions

The following restrictions apply when the Cortex®-M85 processor uses coprocessor instructions:

- The `LDC(2)` or `STC(2)` instructions are not supported. If these are included in software with the <coproc> field set to a value between 0-7 and the coprocessor is present and enabled in the appropriate fields in the CPACR or NSACR, the Cortex®-M85 processor always attempts to take an *Undefined instruction* (UNDEFINSTR) UsageFault exception.
- The processor register fields for data transfer instructions must not include the stack pointer (Rt = `0xD`), this encoding is **UNPREDICTABLE** in the Arm®v8.1-M architecture and results in an UNDEFINSTR UsageFault exception in the Cortex®-M85 processor if the coprocessor is present and enabled in the CPACR or NSACR.
- If any coprocessor instruction is executed when the corresponding coprocessor is either not present or disabled in the CPACR or NSACR, the Cortex®-M85 processor always attempts to take a *No coprocessor* (NOCP) UsageFault exception.

For more information on the CPACR and NSACR, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

13.5 Debug access to coprocessor registers usage constraints

The Cortex®-M85 processor does not support a mechanism to read and write registers located in external coprocessors.

Arm recommends that you implement a coprocessor with a dedicated AHB or APB slave interface for the system to access the registers. If the debug view of the coprocessor is located in the PPB region of the memory map, you can use this interface to connect to the *External Private Peripheral Bus* (EPPB) interface of the Cortex®-M85 processor.

If Secure debug is disabled, you must ensure the Secure information in the coprocessors is protected and not accessible when using a Non-secure debugger.

If the debug slave interface to the coprocessor is connected to the processor *Master AXI* (M-AXI) or *Peripheral AHB* (P-AHB) master interfaces or the EPPB interface, you can use the **ARPROT[1]**, **AWPROT[1]**, **HNONSEC**, and **PPROT[2]** signals on the M-AXI and P-AHB, and APB interfaces respectively. This is because the security level of the debug requests routed through the processor from the D-AHB interface are subject to the debug access and authentication checks.

If the coprocessor state is memory-mapped, then software can also access the information using load and store instructions. If your implementation uses this functionality, you must ensure the appropriate barrier instructions are included to guarantee ordering between coprocessor instructions and load/store operations to the same state.

13.6 Exceptions and context switch

The Cortex®-M85 processor does not include support for automatic save and restore of coprocessor registers on entry and exit to exceptions, unlike the internal processor integer and floating-point registers. Any coprocessor state that must be maintained across a context switch must be carried out by the software that is aware of the coprocessor requirements.

You must ensure that when the coprocessor contains Secure data, it cannot be accessed by software running in a Non-secure exception handler.

13.7 Response to coprocessor errors

The coprocessor must not rely on a synchronous exception that is taken when asserting a CPERROR response to a coprocessor transaction, because the UNDEFINSTR UsageFault might be preempted by a higher priority interrupt in the Cortex®-M85 processor. There is no guarantee that there are no side effects from the erroneous instruction.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

14 Floating-point and MVE support

This chapter describes the *Extension Processing Unit* (EPU), which controls floating-point and *M-profile Vector Extension* (MVE) support.

14.1 Floating-point and MVE operation

The *Extension Processing Unit* (EPU) can be configured to perform floating-point and *M-profile Vector Extension* (MVE) operations.

Scalar floating-point operation

The Cortex®-M85 processor can be configured to provide scalar half, single, and double-precision floating-point operation. The floating-point operation is an implementation of the scalar half, single, and double-precision variants of the Floating-point Extension, FPv5 architecture. Configuring the processor to include floating-point supports all half, single, and double-precision data-processing instructions and data types described in the *Arm®v8-M Architecture Reference Manual*.

The processor supports scalar half, single, and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. The floating-point functionality that the processor supports also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

M-profile Vector Extension operation

The Cortex®-M85 processor can be configured to provide MVE operation. The MVE functionality that is supported depends on the inclusion of floating-point functionality.

- If floating-point functionality is not included, the processor can be configured to any of the following:
    - Not include MVE.
    - Include the integer subset of MVE only (MVE-I). MVE-I operates on 8-bit, 16-bit, and 32-bit data types.
- If floating-point functionality is included, the processor can be configured to any of the following:
    - Not include MVE.
    - Include the integer, half-precision, and single-precision floating-point MVE (MVE-F). MVE-F operates on half-precision and single-precision floating-point values. MVE-F also includes support for MVE-I.

Vector instructions operate on a fixed vector width of 128 bits. The lane width of an operation to be performed is specified by the instruction that is being executed. And an element refers to the data that is put into a lane. Multiple lanes can be executed per beat. There are four beats per vector instruction.

For more information on the MVE extension and terminology, see *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- The Cortex®-M85 processor provides floating-point computation functionality included with the MVE and Floating-point Extension, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*.
- The scalar Floating-point Extension can be implemented with or without *M-profile Vector Extension - floating-point* (MVE-F).

14.1.1 EPU views of the register bank

The *Extension Processing Unit* (EPU) provides an extension register file with registers that can be viewed as:

- Thirty-two 32-bit single-word registers, S0-S31.
- Sixteen 64-bit doubleword registers, D0-D15.
- Eight 128-bit vector registers, Q0-Q7.
- A combination of registers from these views.

14.1.2 Modes of operation

The Cortex®-M85 processor supports the following modes of operation:

- Flush to-zero
- Half-precision flush to-zero
- Default NaN

For more information on these modes, see the *Arm®v8-M Architecture Reference Manual*.

14.1.3 Compliance with the IEEE 754 standard

The Cortex®-M85 processor provides floating-point computation functionality included with the MVE and Floating-point Extension, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*. No support code is required to achieve this compliance.

14.1.4 Exceptions

The *Extension Processing Unit* (EPU) sets the cumulative exception status flags in the FPSCR register as required for each instruction, in accordance with the FPv5 architecture. The EPU does not support exception traps.

The processor also has six output pins, each pin reflects the status of one of the cumulative exception flags:

- Inexact result.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The input is denormal.
- Overflow.
- Underflow.
- Divide-by-zero.
- Invalid operation

14.2 Floating-point and MVE register summary

The *Extension Processing Unit* (EPU) has various registers that support floating-point and *M-profile Vector Extension* (MVE) operations.

The following table shows a summary of the floating-point registers. These registers are described in the *Arm®v8-M Architecture Reference Manual*.

Note

FPCCR, FPCAR, and FPDSCR are banked between Security states.

Table 14-1: Floating-point and MVE register summary

Address	Name	Type	Reset value	Description
`0xE000EF34`	FPCCR	RW	`0xC0000004`	Floating-point Context Control Register (S)
`0xE000EF38`	FPCAR	RW	`0x00000000`	Floating-point Context Address Register (S)
`0xE000EF3C`	FPDSCR	RW	See 14.3 FPDSCR and FPSCR register reset values on page 226	Floating-point Default Status Control Register (S)
This register is not memory mapped	FPSCR	RW		Floating-point Status and Control Register
`0xE000EF40`	MVFR0	RO	Table 5-3: MVFR0, MVFR1, and MVFR2 reset values on page 56	Media and VFP Feature Register 0
`0xE000EF44`	MVFR1	RO		Media and VFP Feature Register 1
`0xE000EF48`	MVFR2	RO		Media and VFP Feature Register 2

14.3 FPDSCR and FPSCR register reset values

The following table shows the reset values for *Floating-point Default Status Control Register* (FPDSCR) and *Floating-point Status and Control Register* (FPSCR) depending on inclusion and exclusion of floating-point and *M-profile Vector Extension* (MVE) functionality.

Table 14-2: FPDSCR and FPSCR reset values

Register name	Reset value	Floating-point and MVE configuration
FPDSCR	**RES0**	Floating-point and MVE are not included.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Register name	Reset value	Floating-point and MVE configuration
	0x00040000	Scalar half, single, and double-precision floating-point is included. MVE is not included.
		Floating-point is not included. Integer subset of MVE is included.
		Scalar half, single, and double-precision floating-point is included. Integer and half and single-precision floating-point MVE is included.
FPSCR	**RES0**	Floating-point and MVE are not included.
	0x00040000	Scalar half, single, and double-precision floating-point is included. MVE is not included.
	0x00040000	Floating-point is not included. Integer subset of MVE is included.
	0x00040000	Scalar half, single, and double-precision floating-point is included. Integer and half and single-precision floating-point MVE is included.

14.4 Powering down the EPU

Depending on your implementation, the *Extension Processing Unit* (EPU) can be in a separate power domain, PDEPU. The way the EPU power domain is powered down depends on whether the EPU domain includes state retention logic.

For more information on powering down the EPU, see 7.7 PDEPU low-power requirements on page 126.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

15 Debug

This chapter describes the debug system.

15.1 Debug functionality

The Cortex®-M85 processor debug functionality includes Arm®v8-M, Arm®v8.1-M, and CoreSight™ features that are designed to support debug and trace of software running on the processor.

These features include:

- A *BreakPoint Unit* (BPU) which can be configured to support four or eight hardware breakpoints.
- A *Data Watchpoint and Trace* (DWT) unit which can be configured to support four or eight hardware comparators that can match both address and data values.
- Support for the *Digital Signal Processing* (DSP) debug extension for analysis of signal processing and compute-based software.
- Monitor mode exception for self-hosted debug.
- Full access to the memory map and registers through a 32-bit *Debug AHB* (D-AHB) interface.
- An *Instrumentation Trace Macrocell* (ITM) for software-driven `printf` debugging which can be linked to the DWT.
- An implementation of the *Performance Monitoring Unit* (PMU).
- An *Embedded Trace Macrocell* (ETM) which supports complete instruction trace. It implements the ETMv4.5 architecture, including support for tracing the *M-profile Vector Extension* (MVE) features. Data trace is not supported. For more information on the ETM, see the *Arm® CoreSight™ ETM-M85 Technical Reference Manual*.
- Access control that prevents unauthorized debug or trace of Secure state or memory, including support for the Unprivileged Debug Extension for fine-grain control of debug access to the processor.

Note

- All trace functionality on the Cortex®-M85 processor is optional.
- The debugger cannot write to the *Interrupt Program Status Register* (IPSR).
- The Cortex®-M85 processor is also supplied with an optional *Trace Port Interface Unit* (TPIU). For more information, see B Trace Port Interface Unit on page 313.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

15.1.1 CoreSight™ discovery

Arm recommends that a debugger identifies and connects to the debug components using the CoreSight™ debug infrastructure.

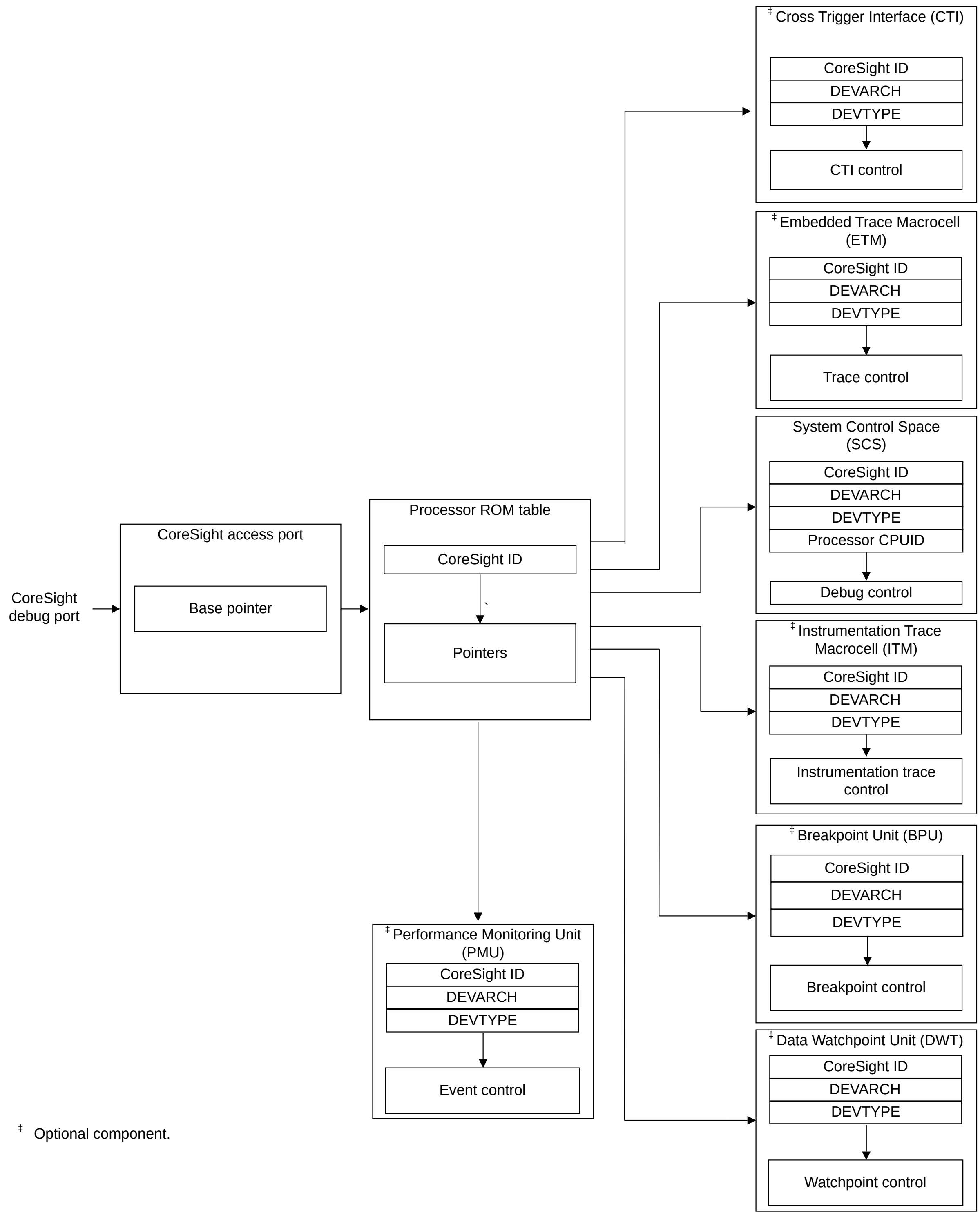
See the *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* for more information.

Arm recommends that a debugger follows the flow in the following figure to discover the components present in the CoreSight™ debug infrastructure. In this case, for each CoreSight™ component in the CoreSight™ system, a debugger reads:

- The peripheral and component ID registers.
- The DEVARCH and DEVTYPE registers.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 15-1: CoreSight™ discovery



To identify the Cortex®-M85 processor and debug components within the CoreSight™ system, Arm recommends that a debugger performs the following actions:

1. Locate and identify the Cortex®-M85 processor ROM table using its CoreSight™ identification.
2. Follow the pointers in the Cortex®-M85 processor ROM table to identify the presence of the following components:
    a. *Cross Trigger Interface* (CTI)
    b. *Embedded Trace Macrocell* (ETM)
    c. *System Control Space* (SCS)
    d. *Instrumentation Trace Macrocell* (ITM)
    e. *BreakPoint Unit* (BPU)
    f. *Data Watchpoint and Trace* (DWT) unit
    g. *Performance Monitoring Unit* (PMU)

15.1.2 Debugger actions for identifying the processor

When a debugger identifies the *System Control Space* (SCS) from its CoreSight™ identification, it can identify the processor and its revision number from the CPUID register in the SCS at address `0xE000ED00`.

A debugger cannot rely on the Cortex®-M85 processor ROM table being the first ROM table encountered. One or more system ROM tables might be included between the access port and the processor ROM table if other CoreSight™ components are in the system. If a system ROM table is present, it can include a unique identifier for the implementation.

15.1.3 Processor ROM table identification and entries

The ROM table identification registers and its values that the following table shows allow debuggers to identify the processor and its debug capabilities.

The following table shows the CoreSight™ components that the Cortex®-M85 processor ROM table points to.

Table 15-1: Cortex®-M85 processor ROM table components

Address	Component	Reset value	Description
`0xE00FF000`	*System Control Space* (SCS)	`0xFFF0F003`	See 15.1.4 Debug identification block register summary on page 233
`0xE00FF004`	*Data Watchpoint and Trace* (DWT)	`0xFFF02003`	See 18 Data Watchpoint and Trace unit on page 258
`0xE00FF008`	*BreakPoint Unit* (BPU)	`0xFFF03003`	See 20 BreakPoint Unit on page 298

Address	Component	Reset value	Description
0xE00FF00C	*Instrumentation Trace Macrocell* (ITM)	• If ITM is implemented, 0xFFF01003. • If ITM is not implemented, 0xFFF01002.	See 17 Instrumentation Trace Macrocell on page 251
0xE00FF010	**TRACE PORT INTERFACE UNIT** (TPIU)	0xFFF41002	The TPIU is not configured inside the processor. It can be configured in the MCU layer and included in the MCU ROM table See B Trace Port Interface Unit on page 313
0xE00FF014	*Embedded Trace Macrocell* (ETM)	• If ETM is implemented, 0xFFF42003. • If ETM is not implemented, 0xFFF42002.	See the *Arm® CoreSight™ ETM-M85 Technical Reference Manual*
0xE00FF018	*Performance Monitoring Unit* (PMU)	0xFFF04003	See 17 Instrumentation Trace Macrocell on page 251
0xE00FF01C	*Cross Trigger Interface* (CTI)	0xFFF43003	See 19 Cross Trigger Interface on page 266
0xE00FF020 - 0xE00FFFC8	Reserved	-	-
0xE00FFFCC	SYSTEM ACCESS	0x00000001	See the *Arm® CoreSight™ Architecture Specification v3.0*
0xE00FFFD0 - 0xE00FFFEC	Peripheral ID registers	Table 15-2: Cortex-M85 processor ROM table identification values on page 232.	
0xE00FFFF0 - 0xE00FFFC	Component ID registers		

The Cortex®-M85 processor ROM table entries point to the debug components of the processor. The offset for each entry is the offset of that component from the ROM table base address, 0xE00FF000.

See the *Arm® CoreSight™ Architecture Specification v3.0* for more information about the ROM table ID and component registers, and access types.

Table 15-2: Cortex®-M85 processor ROM table identification values

Address	Name	Type	Reset value	Description
0xE00FFFD0	PIDR4	RO	0x00000004	See *Arm®v8-M Architecture Reference Manual* for more information.
0xE00FFFD4	PIDR5	RO	0x00000000	
0xE00FFFD8	PIDR6	RO	0x00000000	
0xE00FFFDC	PIDR7	RO	0x00000000	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE00FFFE0	PIDR0	RO	0x000000D4	
0xE00FFFE4	PIDR1	RO	0x000000B4	
0xE00FFFE8	PIDR2	RO	0x0000000B	
0xE00FFFEC	PIDR3	RO	0x00000000	
0xE00FFFF0	CIDR0	RO	0x0000000D	
0xE00FFFF4	CIDR1	RO	0x00000010	
0xE00FFFF8	CIDR2	RO	0x00000005	
0xE00FFFFC	CIDR3	RO	0x000000B1	

These values for the Peripheral ID registers identify this as the Cortex®-M85 processor ROM table. The Component ID registers identify this as a CoreSight™ ROM table.

Note

The Cortex®-M85 processor ROM table only supports word-size transactions.

15.1.4 Debug identification block register summary

The *System Control Space* (SCS) provides a set of debug identification registers which can be used for debug-related peripheral and component identification.

The following table shows the debug identification registers and values for debugger detection. For more information, see the *Arm®v8-M Architecture Reference Manual*.

Table 15-3: Debug identification values

Address offset	Name	Type	Reset value	Description
0xE000EFD0	DPIDR4	RO	0x00000004	SCS Peripheral Identification Register 4
0xE000EFD4	DPIDR5	RO	0x00000000	SCS Peripheral Identification Register 5
0xE000EFD8	DPIDR6	RO	0x00000000	SCS Peripheral Identification Register 6
0xE000EFDC	DPIDR7	RO	0x00000000	SCS Peripheral Identification Register 7
0xE000EFE0	DPIDR0	RO	0x00000023	SCS Peripheral Identification Register 0
0xE000EFE4	DPIDR1	RO	0x000000BD	SCS Peripheral Identification Register 1
0xE000EFE8	DPIDR2	RO	0x0000000B	SCS Peripheral Identification Register 2

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address offset	Name	Type	Reset value	Description
`0xE000EFEC`	DPIDR3	RO	`0x00000000` **Note:** Bits [7:4] and [3:0] are REVAND and CMOD respectively. The REVAND field indicates minor errata fixes specific to this design, for example metal fixes after implementation. If the component is reusable IP, the CMOD field indicates whether you have modified the behavior of the component. These values depend on the exact revision of the silicon as documented in *Arm® CoreSight™ Architecture Specification v3.0*.	SCS Peripheral Identification Register 3
`0xE000EFF0`	DCIDR0	RO	`0x0000000D`	SCS Component Identification Register 0
`0xE000EFF4`	DCIDR1	RO	`0x00000090`	SCS Component Identification Register 1
`0xE000EFF8`	DCIDR2	RO	`0x00000005`	SCS Component Identification Register 2
`0xE000EFFC`	DCIDR3	RO	`0x000000B1`	SCS Component Identification Register 3
`0xE000EFBC`	DDEVARCH	RO	`0x47702A04`	SCS Device Architecture Register
`0xE000EFCC`	DDEVTYPE	RO	`0x00000000`	SCS Device Type Register

15.1.5 Debug register summary

The following table shows the debug registers, with address, name, type, reset value, and description information for each register.

Each register is 32-bits wide. These registers are not banked between Security states or are banked between Security states on a bit by bit basis. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*

Table 15-4: Debug register summary

Address	Name	Type	Reset value	Description
`0xE000ED30`	DFSR	RW	`0x00000000` Cold reset only.	Debug Fault Status Register
`0xE000EDF0`	DHCSR	RW	`0x02000000`	Debug Halting Control and Status Register
`0xE000EDF4`	DCRSR	WO	`0xXXXX00XX`, bits [15:7] are **RES0**	Debug Core Register Selector Register
`0xE000EDF8`	DCRDR	RW	**UNKNOWN**	Debug Core Register Data Register
`0xE000EDFC`	DEMCR	RW	`0x00000000`	Debug Exception and Monitor Control Register
`0xE000EE04`	DAUTHCTRL	RW	`0x00000000`	Debug Authentication Control Register
`0xE000EE08`	DSCSR	RW	`0x00030000`	Debug Security Control and Status Register

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE000EFB8`	DAUTHSTATUS	RO	`0x00XX00XX`	Debug Authentication Status Register

15.2 D-AHB interface

The 32-bit *Debug AHB* (D-AHB) interface implements the AMBA® 5 AHB protocol. It can be used with a CoreSight™ AHB-AP to provide debugger access to all processor control and debug resources, and a view of memory that is consistent with that observed by load and store operations.

Accesses on the D-AHB interface are always little-endian.

Debugger accesses are distributed to the appropriate internal and external resource according to the address of the request. Accesses on the D-AHB are reflected on the TCM, *Master AXI* (M-AXI), *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) as appropriate.

15.2.1 Debug memory access

The Cortex®-M85 processor implements external debug interaction through a 32-bit AMBA® 5 AHB debug interface.

This interface can be integrated with a suitable CoreSight™ AHB-AP interface and provides debugger access to:

- All processor control and debug resources.
- A view of memory, which is consistent with the view that software load and store operations observe.

Accesses on the D-AHB interface always ignore the endianness attribute and do not pass through the data swizzling logic in the processor used for load and store requests. Therefore, accesses to addresses outside the PPB region observe data in the downstream memory endian format and accesses in the PPB region observe data in little-endian format.

- *Debug AHB* (D-AHB) accesses undergo security attribution and security access checks. The debug Security state depends on DHCSR.S_SDE and the D-AHB input signal, **HNONSECD**, which indicates the security level that a debug access requests. If this signal is asserted, this indicates that the transfer is Non-secure.
- D-AHB accesses are not checked against the *Memory Protection Unit* (MPU) for memory attribute checks unless the Unprivileged Debug is enabled for a debug Security state.
  - Unprivileged Debug is enabled for the secure debug state when DHCSR.S_SUIDE is set.
  - Unprivileged Debug is enabled for the Non-secure debug state when DHCSR.S_NSUIDE is set.
- If unprivileged debug is enabled, then the access is always treated as unprivileged, regardless of the value of the D-AHB signal bit **HPROTD[1]** and reported on the D-AHB interface.
  - If the debug Security state is Secure, then the D-AHB access is subject to permission checks based on regions that are defined in the Secure MPU.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - If the debug Security state is Non-secure, then the D-AHB access is subject to permission checks based on regions that are defined in the Non-secure MPU.

- D-AHB accesses to the EPPB memory region (`0xE0040000-0xE00FEFFF`) must be marked as privileged, **HPROT[1]** HIGH, unless unprivileged invasive *Debug Access Port* (DAP) access is enabled by setting DAUTHCTRL.UIDAPEN for the debug security state. When DAUTHCTRL.UIDAPEN is set all the peripherals in the EPPB region can be accessed by non-privileged debug accesses through D-AHB except for the:
    - *External Wakeup Interrupt Controller* (EWIC) located at `0xE0047000-0xE0047FFF`.
    - Reserved regions at `0xE0046000-0xE0046FFF` and `0xE0048000-0xE0048FFF`

  These regions can only be accessed with Secure privileged requests. Any non-privileged accesses returns an error on D-AHB.

- D-AHB accesses to the internal PPB region must be marked as privileged, unless unprivileged invasive DAP access is enabled by setting DAUTHCTRL.UIDAPEN for the debug Security state.
    - When DAUTHCTRL.UIDAPEN is set, many of the registers in the internal PPB region can be accessed. The exceptions are those registers which are normally accessible by unprivileged code. For example, some of the *Instrumentation Trace Macrocell* (ITM) registers and the STIR. For more information on the ITM registers, see 17.2 ITM register summary on page 252. For more information on STIR, see *Arm®v8-M Architecture Reference Manual*.
    - When DAUTHCTRL.UIDAPEN is not set and the debug access is unprivileged, then almost all accesses to the PPB registers get an error response. However, the registers which are normally accessible by unprivileged code cannot be accessed. For example, some of the *Instrumentation Trace Macrocell* (ITM) registers and the STIR. For more information on the ITM registers, see 17.2 ITM register summary on page 252. For more information on STIR, see *Arm®v8-M Architecture Reference Manual*.

- The security of a debug transaction on one of the external interfaces is determined by all of the following:
    - The access control signals.
    - The mapping of the address in the *Security Attribution Unit* (SAU) and *Implementation Defined Attribution Unit* (IDAU).
    - The internal debug state of the processor.
    - The **HNONSECD** signal value that is associated with the D-AHB debug request.

Note

- For more information on the DHCSR and DAUTHCTRL registers, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on all the AMBA® 5 AHB-compliant D-AHB signals mentioned in this section, see the *Arm® AMBA® 5 AHB Protocol Specification*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

15.2.2 Debugger access memory attributes and data cache access

The memory attributes associated with debugger accesses on *Debug AHB* (D-AHB) depend on the debug access mode.

Unprivileged Debug is not enabled

If Unprivileged Debug is not enabled, debugger accesses are not subject to the memory attributes defined by the *Memory Protection Unit* (MPU). Instead, the memory attributes used to perform a debugger access are derived from the **HPROTD** signal on D-AHB. The attributes are used differently depending on the memory region that is associated with the address.

The following table shows the behavior of debug accesses and dependency on **HPROTD** for both internal and externally memory-mapped regions when Unprivileged Debug is not enabled.

Table 15-5: HPROTD attributes

Region and interface	Description
CODE and SRAM regions TCM and *Master AXI* (M-AXI) interfaces	**Accesses to ITCM and DTCM** **HPROTD[1]** is passed through to **ITCMPRIV** and **DTCMPRIV**. **HPROTD[0]** is ignored. **ITCMMASTER** and **DTCMMASTER** signals are asserted indicating a debugger access. **Accesses to M-AXI** If an access is not completed in the data cache: • **HPROTD[0]** is ignored. All debugger accesses are performed with **ARPROT[2]** and **AWPROT[2]** set to 0. • **HPROTD[6:1]** is passed through to **ARPROT[0]**, **AWPROT[0]**, **ARCACHE**, and **AWCACHE**. **ARMASTER** and **AWMASTER** are asserted indicating a debugger access.
Peripheral, external RAM/Device, Vendor_SYS regions M-AXI and *Peripheral AHB* (P-AHB) interfaces	**Accesses to P-AHB** **HPROTD[0]** is ignored. All debugger accesses are performed with **HPROTP[0]** set to 1. **HPROTD[6:1]** is passed to P-AHB. **HMASTERP** is asserted indicating debugger access. **Accesses to M-AXI** If an access is not completed in the data cache: • **HPROTD[0]** is ignored. All debugger accesses are performed with **ARPROT[2]** and **AWPROT[2]** set to 0. • **HPROTD[6:1]** is passed through to **ARPROT[0]**, **AWPROT[0]**, **ARCACHE**, and **AWCACHE**. **ARMASTER** and **AWMASTER** are asserted indicating a debugger access. **Note:** The debugger access can complete in the data cache if the software has programmed the MPU to make this region cacheable.
Internal Private Peripheral Bus (IPPB)	• **HPROTD[0]** is ignored. • **HPROTD[1]** is used for register-specific checks. • **HPROTD[6:2]** is ignored. Unprivileged D-AHB accesses to privileged registers return an ERROR response on **HRESPD**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Region and interface	Description
CORE or **DEBUG** *External Private Peripheral Bus* (EPPB)	• **HPROT[0]** is ignored. • **HPROT[1]** is passed through to **PPROT[0]**. • **PADDR31** is asserted which indicates a debugger access.

All debug read and write accesses marked as Normal cacheable and Non-shareable in **HPROTD** and outside the address regions associated with ITCM and DTCM look up the data cache if it is configured in the processor. If the address is present in the cache, for a read the data is returned without making any request on M-AXI and for a write the cache line is updated. If the debug memory attribute is Write-through, then the data is also be written on M-AXI. Debugger accesses never allocate lines to the cache on a miss. Debug accesses marked as Device, Non-cacheable or Normal shareable in **HPROTD** do not look up the data cache.

Unprivileged Debug is enabled

If Unprivileged Debug is enabled, the **HPROTD[6:0]** input signals on D-AHB are ignored and the debugger accesses are always treated as unprivileged. The *Memory Protection Unit* (MPU) determines the memory attributes associated with debugger accesses and debugger accesses are subject to MPU checks.

The following table shows the behavior of debug accesses for both internal and externally memory-mapped regions when Unprivileged Debug is enabled. Accesses that are not allowed will return an error to the D-AHB instead of proceeding to an external interface.

Table 15-6: Memory attributes

Region and interface	Description
CODE and SRAM regions TCM and *Master AXI* (M-AXI) interfaces	**Accesses to ITCM and DTCM** **ITCMPRIV** and **DTCMPRIV** are forced to 0. **ITCMMASTER** and **DTCMMASTER** signals are asserted indicating a debugger access. **Accesses to M-AXI** If an access is not completed in the data cache: • All debugger accesses are performed with **ARPROT[2]** and **AWPROT[2]** set to 0. • All debugger accesses are performed with **ARPROT[0]** and **AWPROT[0]** set to 0. • The memory attributes associated with the address in the MPU are passed to **ARCACHE** and **AWCACHE**. • **ARMASTER** and **AWMASTER** are asserted indicating a debugger access.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

<table>
<tr><th>Region and interface</th><th>Description</th></tr>
<tr><td>Peripheral, external RAM/Device, Vendor_SYS regions
M-AXI and Peripheral AHB (P-AHB) interfaces</td><td>Accesses to P-AHB
All debugger accesses are performed with HPROTP[0] set to 1.
The memory attributes are passed to P-AHB.
HMASTERP is asserted indicating debugger access.
Accesses to M-AXI
If an access is not completed in the data cache:
• All debugger accesses are performed with ARPROT[2] and AWPROT[2] set to 0.
• All debugger accesses are performed with ARPROT[0] and AWPROT[0] set to 0.
• The memory attributes associated with the address in the MPU are passed to ARCACHE and AWCACHE.
• ARMASTER and AWMASTER are asserted indicating a debugger access.</td></tr>
<tr><td>Internal Private Peripheral Bus (IPPB)</td><td>Unprivileged access in some registers is allowed when DAUTHCTRL.UIDAPEN is set. if Unprivileged access is not allowed, an error response is returned on HRESPD.</td></tr>
<tr><td>CORE or DEBUG External Private Peripheral Bus (EPPB)</td><td>COREPADDR31 or PADDR31 is asserted which indicates a debugger access.</td></tr>
</table>

15.2.3 Debug access security and attributes

Debugger accesses to memory and any memory-mapped registers are subject to the same security checks as data accesses generated by software running on the processor, with the security attributes set as the following:

- Request is Secure if the DHCSR.S_SDE register field is 1 indicating secure debug is enabled and **HNONSECD** is LOW.
- Otherwise the request is Non-secure.

The state of DHCSR.S_SDE depends on the context of the debug request. If the processor is halted when it was in Secure state, then DHCSR.S_SDE is 1, otherwise the value of the field depends on the secure access control input signal. This implies access to the secure state and memory is only available if secure invasive debug is permitted in the system.

In most of the memory regions, debugger accesses are subject to validation and attribution. This implies that the final security state of an access on the *Master AXI* (M-AXI), *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) interfaces are set by the *Security Attribution Unit* (SAU) in the same way as software generated accesses. The SAU blocks memory accesses which do not have the required permissions. For example, accesses to memory regions marked as Secure in the SAU if DHCSR.S_SDE is 0 or **HNONSECD** is HIGH. This results in an error response on the *Debug AHB* (D-AHB) interface, but unlike accesses originating from software, a SecureFault is not raised.

There are a number of address regions associated with the *System Control Space* (SCS) and debug peripherals where the security state of the access is determined only by the **HNONSECD** signal and DHCSR.S_SDE.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

For more information on the DHCSR register, see the *Arm®v8-M Architecture Reference Manual*. For more information on all the AMBA® 5 AHB-compliant **HNONSECD** signal, see the *Arm® AMBA® 5 AHB Protocol Specification*.

15.2.4 Debug during reset and before code execution commences

The Cortex®-M85 processor supports access to the debug and trace resource from a debug agent connected to the *Debug AHB* (D-AHB) interface when the device is in processor reset. This can be useful for setting up the debug and trace environment before any code has executed on the processor.

The following table lists the memory regions which can be accessed during processor reset. Access control and security level are determined in the same manner as debug accesses during code execution or when halted based on the authentication signals and the default SAU/IDAU regions. Any component on the EPPB, which cannot be accessed during reset, must ensure the APB **PREADY** signal is HIGH in response to a request from the processor.

Note

The Cortex®-M85 D-AHB interface is in the PDDEBUG power domain. The debugger is able to access CoreSight™ components in PDDEBUG when PDDEBUG is powered on and a debugger connection is established. This includes the ITM, DWT, BPU, PMU, ETM, CTI, Processor ROM table, andDebug EPPB APB interface.

Access to all other memory areas during processor reset is **UNPREDICTABLE**.

Table 15-7: Debug and trace registers accessible during processor reset

Memory address range	Group	Description
`0xE000E004`	System Control and ID registers	ICTR register. For more information on the ICTR register, see the *Arm®v8-M Architecture Reference Manual*
`0xE000ECFC`		REVIDR register. 5.6 REVIDR, Revision ID Register on page 63.
`0xE000ED00`		CPUID register. 5.4 CPUID, CPUID Base Register on page 58.
`0xE000ED30`		DFSR register. For more information on the DFSR register, see 5.1 System control register summary on page 49.
`0xE000ED40-0xE000ED7F`		ID registers. 5.2 Identification register summary on page 52.
`0xE000ED80`		CCSIDR register. 5.5.3 CCSIDR, Current Cache Size ID Register on page 62

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Memory address range	Group	Description
0xE000EDF0-0xE000EEFF		Debug registers. 15.1.5 Debug register summary on page 234.
0xE000EF40-0xE000EF4B		MVFR0, MVFR1, MVFR2 registers. For more information on the MVFR0, MVFR1, MVFR2 registers, see 5.1 System control register summary on page 49.
0xE000EF4B-0xE000EFFF		Debug Identification Block. 15.1.4 Debug identification block register summary on page 233.
0xE0000000-0xE0000FFF	*Instrumentation Trace Macrocell* (ITM)	17.2 ITM register summary on page 252
0xE0001000-0xE0001FFF	*Data Watchpoint and Trace* (DWT)	18.5 DWT register summary on page 262
0xE0002000-0xE0002FFF	*BreakPoint Unit* (BPU)	20.2 BPU register summary on page 298
0xE0003000-0xE0003FFF	*Performance Monitoring Unit* (PMU)	16.3 PMU register summary on page 249
0xE0041000-0xE0041FFF	*Embedded Trace Macrocell* (ETM)	For more information, see the *Arm® CoreSight™ ETM-M85 Technical Reference Manual*
0xE0042000-0xE0042FFF	*Cross Trigger Interface* (CTI)	19.2 CTI register summary on page 268
0xE0044000-0xE00FEFFF	*Debug External Private Peripheral Bus* (Debug EPPB)	Access directed to Cortex®-M85 Core EPPB APB interface.
0xE00FF000-0xE00FFFFF	Processor ROM table	-

An alternative way to access debug resources before code execution is to use the **CPUWAIT** input signal. If this signal is asserted after reset, the processor will delay execution of any instructions until it is de-asserted. During this time, debug accesses can be carried out over the D-AHB interface. This approach has the advantage that the reset has completed, so all registers and memory can be accessed instead of just a restricted set when the reset is active.

15.2.5 Advanced DSP debug capabilities

The Cortex®-M85 processor supports the *Digital Signal Processing* (DSP) Debug Extension to provide additional features for analyzing signal processing and compute software using the *Data Watchpoint and Trace* (DWT) and *Performance Monitoring Unit* (PMU).

For more information on the DSP Debug Extension, see the *Arm®v8-M Architecture Reference Manual* and include the following additional functionality to the processor.

The DSP debug capabilities supported are:

DWT value mask

Value matching using the DWT comparators, DWT_COMPn, is extended to use a mask register DWT_VMASKn. This allows events to be selected based on sub-word values or arbitrary bitfields. This is useful for analyzing data where only part of the data word is valid.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Halt request on PMU overflow

The processor can be configured to enter debug Halt when a PMU counter, which is configured to generate an interrupt overflow. This can be used to set up a hardware watchpoint which is triggered after a number of events have been observed in a system.

Extended PMU events

The DSP Debug Extension defines additional PMU events specific to M-profile debug and trace operation TRCEXTOUT, CTI_TRIGOUT and DWT_CMPMATCH. For more information on these events, see 16.2 PMU events on page 243.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

16 Performance Monitoring Unit Extension

This chapter describes the *Performance Monitoring Unit* (PMU) Extension.

16.1 PMU features

The Cortex®-M85 processor *Data Watchpoint and Trace* (DWT) implements the *Performance Monitoring Unit* (PMU). This enables software to get information about events that are taking place in the processor and can be used for performance analysis and system debug.

The PMU supports eight 16-bit event counters and one 32-bit cycle counter. Each event counter can count one event from a list comprising both architectural and **IMPLEMENTATION DEFINED** events. For more information on PMU events, see 16.2 PMU events on page 243. The PMU also supports a chain function which allows the PMU to cascade two of the 16-bit counters into one 32-bit counter. Only odd event counters support the chain feature. PMU counters increment if the appropriate bit in PMU_CNTENSET register is set.

The Arm®v8.1-M architecture specifies that operation of the PMU counters and DWT profiling counters is mutually exclusive. The Cortex®-M85 processor uses this requirement to share the state used for the counters.

The PMU cycle counter PMU_CCNTR is an alias of the DWT_CYCCNT register. All derived functions of the counter are available whenever either the DWT or the PMU enables the cycle counter. DWT_CTRL.NOCYCCNT is RAZ.

Generating interrupts

If a counter is configured to generate an interrupt when it overflows, DEMCR.MON_PEND is set to 1 to make a Debug Monitor exception pended with DFSR.PMU set to 1. The associated overflow bit programmed by PMU_OVSSET and PMU_OVSCLR indicates which counter triggered the exception. The interrupts are enabled if their corresponding bit programmed by PMU_INTENSET and PMU_INTENCLR is set and DEMCR.MON_EN is 1.

Exporting trace

The PMU can export trace whenever the lower 8 bits of the counters overflow. The PMU issues an event counter packet with the appropriate counter flag set to 1. This occurs on counter increment only, not on software or debugger write. For each counter n, if the lower 8 bits of that counter overflows, the associated OVn bit of the event counter packet is set. If multiple counters overflow during the same period, multiple bits might be set.

The PMU can serve as an event source for the *Cross Trigger Interface* (CTI).

For more information on the registers mentioned in this section, see the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

16.2 PMU events

The following table shows the events that are generated and the numbers that the *Performance Monitoring Unit* (PMU) uses to reference the events.

Table 16-1: PMU events

Event number	Event mnemonic	PMU event bus bit	Event name
`0x0000`	SW_INCR	0	Instruction architecturally executed, condition code check pass, software increment
`0x0001`	L1I_CACHE_REFILL	1	L1 instruction cache linefill
`0x0003`	L1D_CACHE_REFILL	2	L1 data cache linefill
`0x0004`	L1D_CACHE	3	L1 data cache access
`0x0006`	LD_RETIRED	4	Instruction architecturally executed, condition code check pass, load
`0x0007`	ST_RETIRED	5	Instruction architecturally executed, condition code check pass, store
`0x0008`	INST_RETIRED	6	Instruction architecturally executed
`0x0009`	EXC_TAKEN	7	Exception taken
`0x000A`	EXC_RETURN	8	Instruction architecturally executed, condition code check pass, exception return
`0x000C`	PC_WRITE_RETIRED	9	Instruction architecturally executed, condition code check pass, software change of the PC
`0x000D`	BR_IMMED_RETIRED	10	Instruction architecturally executed, immediate branch
`0x000E`	BR_RETURN_RETIRED	11	Instruction architecturally executed, condition code check pass, procedure return
`0x000F`	UNALIGNED_LDST_RETIRED	12	Instruction architecturally executed, condition code check pass, unaligned load or store
`0x0010`	BR_MIS_PRED	13	Mis-predicted or not predicted branch speculatively executed
`0x0011`	CPU_CYCLES	14	Cycle
`0x0012`	BR_PRED	15	Predictable branch speculatively executed
`0x0013`	MEM_ACCESS	16	Data memory access
`0x0014`	L1I_CACHE	17	L1 instruction cache access
`0x0015`	L1D_CACHE_WB	18	L1 data cache write-back
`0x0019`	BUS_ACCESS	19	Any beat access to the M-AXI read interface, M-AXI write interface and any access to P-AHB interface or *External Private Peripheral Bus* (EPPB) interface
`0x001A`	MEMORY_ERROR	20	ECC error for *Tightly Coupled Memories* (TCMs) and caches
`0x001D`	BUS_CYCLES	22	AXI Bus cycle when ACKEN is asserted
`0x001E`	CHAIN	23	For an odd-numbered counter, increments when an overflow occurs on the preceding even-numbered counter on the same PE
`0x0021`	BR_RETIRED	25	Instruction architecturally executed, branch
`0x0022`	BR_MIS_PRED_RETIRED	26	Instruction architecturally executed, mispredicted branch

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Event number	Event mnemonic	PMU event bus bit	Event name
0x0023	STALL_FRONTEND	27	If there are no instructions available from the fetch stage of the processor pipeline, the processor considers the front-end of the processor pipeline as being stalled
0x0024	STALL_BACKEND	28	If there is an instruction available from the fetch stage of the pipeline but it cannot be accepted by the decode stage of the processor pipeline, the processor considers the back-end of the processor pipeline as being stalled
0x0036	LL_CACHE_RD	29	L1 data cache read For the Cortex®-M85 processor, this event is the same as L1D_CACHE_RD
0x0037	LL_CACHE_MISS_RD	30	L1 data cache read miss For the Cortex®-M85 processor, this event is the same as L1D_CACHE_MISS_RD
0x0039	L1D_CACHE_MISS_RD	31	L1 data cache read miss For the Cortex®-M85 processor, this event is the same as LL_CACHE_MISS_RD
0x003C	STALL	34	No operation sent for execution
0x0040	L1D_CACHE_RD	38	L1 data cache read For the Cortex®-M85 processor, this event is the same as LL_CACHE_RD
0x0100	LE_RETIRED	39	Loop end instruction architecturally executed, entry registered in the LO_BRANCH_INFO cache
0x0108	LE_CANCEL	43	LO_BRANCH_INFO cache containing a valid loop entry cleared while not in the last iteration of the loop
0x0114	SE_CALL_S	45	Call to secure function, resulting in security state change
0x0115	SE_CALL_NS	46	Call to Non-secure function, resulting in security state change
0x0118	DWT_CMPMATCH0	47	*Data Watchpoint and Trace* (DWT) comparator 0 match
0x0119	DWT_CMPMATCH1	48	DWT comparator 1 match
0x011A	DWT_CMPMATCH2	49	DWT comparator 2 match
0x011B	DWT_CMPMATCH3	50	DWT comparator 3 match
0x011C	DWT_CMPMATCH4	141	DWT comparator 4 match
0x011D	DWT_CMPMATCH5	142	DWT comparator 5 match
0x011E	DWT_CMPMATCH6	143	DWT comparator 6 match
0x011F	DWT_CMPMATCH7	144	DWT comparator 7 match
0x0200	MVE_INST_RETIRED	51	*M-profile Vector Extension* (MVE) instruction architecturally executed
0x0204	MVE_FP_RETIRED	53	MVE floating-point instruction architecturally executed
0x0208	MVE_FP_HP_RETIRED	55	MVE half-precision floating-point instruction architecturally executed

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Event number	Event mnemonic	PMU event bus bit	Event name
0x020C	MVE_FP_SP_RETIRED	57	MVE single-precision floating-point instruction architecturally executed
0x0214	MVE_FP_MAC_RETIRED	59	MVE floating-point multiply or multiply accumulate instruction architecturally executed
0x0224	MVE_INT_RETIRED	61	MVE integer instruction architecturally executed
0x0228	MVE_INT_MAC_RETIRED	63	MVE integer multiply or multiply-accumulate instruction architecturally executed
0x0238	MVE_LDST_RETIRED	65	MVE load or store instruction architecturally executed
0x023C	MVE_LD_RETIRED	67	MVE load instruction architecturally executed
0x0240	MVE_ST_RETIRED	69	MVE store instruction architecturally executed
0x0244	MVE_LDST_CONTIG_RETIRED	71	MVE contiguous load or store instruction architecturally executed
0x0248	MVE_LD_CONTIG_RETIRED	73	MVE contiguous load instruction architecturally executed
0x024C	MVE_ST_CONTIG_RETIRED	75	MVE contiguous store instruction architecturally executed
0x0250	MVE_LDST_NONCONTIG_RETIRED	77	MVE non-contiguous load or store instruction architecturally executed
0x0254	MVE_LD_NONCONTIG_RETIRED	79	MVE non-contiguous load instruction architecturally executed
0x0258	MVE_ST_NONCONTIG_RETIRED	81	MVE non-contiguous store instruction architecturally executed
0x025C	MVE_LDST_MULTI_RETIRED	83	MVE memory instruction targeting multiple registers architecturally executed
0x0260	MVE_LD_MULTI_RETIRED	85	MVE memory load instruction targeting multiple registers architecturally executed
0x0264	MVE_ST_MULTI_RETIRED	87	MVE memory store instruction targeting multiple registers architecturally executed
0x028C	MVE_LDST_UNALIGNED_RETIRED	89	MVE unaligned memory load or store instruction architecturally executed
0x0290	MVE_LD_UNALIGNED_RETIRED	91	MVE unaligned load instruction architecturally executed
0x0294	MVE_ST_UNALIGNED_RETIRED	93	MVE unaligned store instruction architecturally executed
0x0298	MVE_LDST_UNALIGNED_NONCONTIG_RETIRED	95	MVE unaligned non-contiguous load or store instruction architecturally executed
0x02A0	MVE_VREDUCE_RETIRED	97	MVE vector reduction instruction architecturally executed
0x02A4	MVE_VREDUCE_FP_RETIRED	99	MVE floating-point vector reduction instruction architecturally executed
0x02A8	MVE_VREDUCE_INT_RETIRED	101	MVE integer vector reduction instruction architecturally executed
0x02B8	MVE_PRED	102	Cycles where one or more predicated beats architecturally executed
0x02CC	MVE_STALL	103	Stall cycles caused by an MVE instruction
0x02CD	MVE_STALL_RESOURCE	104	Stall cycles caused by an MVE instruction because of resource conflicts
0x02CE	MVE_STALL_RESOURCE_MEM	105	Stall cycles caused by an MVE instruction because of memory resource conflicts
0x02CF	MVE_STALL_RESOURCE_FP	106	Stall cycles caused by an MVE instruction because of floating-point resource conflicts

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Event number	Event mnemonic	PMU event bus bit	Event name
`0x02D0`	MVE_STALL_RESOURCE_INT	107	Stall cycles caused by an MVE instruction because of integer resource conflicts
`0x02D3`	MVE_STALL_BREAK	108	Stall cycles caused by an MVE chain break
`0x02D4`	MVE_STALL_DEPENDENCY	109	Stall cycles caused by MVE register dependency
`0x4007`	ITCM_ACCESS	110	*Instruction Tightly Coupled Memory* (ITCM) access
`0x4008`	DTCM_ACCESS	111	*Data Tightly Coupled Memory* (DTCM) access
`0x4010`	TRCEXTOUT0	112	*Embedded Trace Macrocell* (ETM) external output 0
`0x4011`	TRCEXTOUT1	113	ETM external output 1
`0x4012`	TRCEXTOUT2	114	ETM external output 2
`0x4013`	TRCEXTOUT3	115	ETM external output 3
`0x4018`	CTI_TRIGOUT4	116	*Cross Trigger Interface* (CTI) output trigger 4
`0x4019`	CTI_TRIGOUT5	117	CTI output trigger 5
`0x401A`	CTI_TRIGOUT6	118	CTI output trigger 6
`0x401B`	CTI_TRIGOUT7	119	CTI output trigger 7
`0xC000`	ECC_ERR	120	One or more *Error Correcting Code* (ECC) errors detected
`0xC001`	ECC_ERR_MBIT	121	One or more multi-bit ECC errors detected
`0xC010`	ECC_ERR_DCACHE	122	One or more ECC errors in the data cache
`0xC011`	ECC_ERR_ICACHE	123	One or more ECC errors in the instruction cache
`0xC012`	ECC_ERR_MBIT_DCACHE	124	One or more multi-bit ECC errors in the data cache
`0xC013`	ECC_ERR_MBIT_ICACHE	125	One or more multi-bit ECC errors in the instruction cache
`0xC020`	ECC_ERR_DTCM	126	One or more ECC errors in the *Data Tightly Coupled Memory* (DTCM)
`0xC021`	ECC_ERR_ITCM	127	One or more ECC errors in the *Instruction Tightly Coupled Memory* (ITCM)
`0xC022`	ECC_ERR_MBIT_DTCM	128	One or more multi-bit ECC errors in the DTCM
`0xC023`	ECC_ERR_MBIT_ITCM	129	One or more multi-bit ECC errors in the ITCM
`0xC100`	PF_LINEFILL	130	The prefetcher starts a linefill
`0xC101`	PF_CANCEL	131	The prefetcher stops prefetching
`0xC102`	PF_DROP_LINEFILL	132	A linefill triggered by the prefetcher has been dropped because of a lack of buffering
`0xC200`	NWAMODE_ENTER	133	No-write allocate mode entry
`0xC201`	NWAMODE	134	Write-Allocate store is not allocated into the data cache due to no-write-allocate mode
`0xC300`	SAHB_ACCESS	135	Read or write access on the S-AHB interface to the TCM
`0xC301`	PAHB_ACCESS	136	Read or write access to the P-AHB write interface
`0xC302`	AXI_WRITE_ACCESS	137	Any beat access to M-AXI write interface.
`0xC303`	AXI_READ_ACCESS	138	Any beat access to M-AXI read interface
`0xC400`	DOSTIMEOUT_DOUBLE	139	Denial of Service timeout has fired twice and caused buffers to drain to allow forward progress
`0xC401`	DOSTIMEOUT_TRIPLE	140	Denial of Service timeout has fired three times and blocked the LSU to force forward progress

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Event number	Event mnemonic	PMU event bus bit	Event name
0xC500	FUSED_INST_RETIRED	145	Fused instructions architecturally executed
0xC501	BR_INDIRECT	146	Indirect branch instruction architecturally executed
0xC502	BTAC_HIT	147	BTAC branch predictor hit
0xC503	BTAC_HIT_RETURNS	148	Return branch hits BTAC
0xC504	BTAC_HIT_CALLS	149	Call branch hits BTAC
0xC505	BTAC_HIT_INDIRECT	150	Indirect branch hits BTACT
0xC506	BTAC_NEW_ALLOC	151	New allocation to BTAC
0xC507	BR_IND_MIS_PRED	152	Indirect branch mis-predicted
0xC508	BR_RETURN_MIS_PRED	153	Return branch mis-predicted
0xC509	BR_BTAC_OFFSET_OVERFLOW	154	Branch does not allocate in BTAC due to offset overflow
0xC50A	STB_FULL_STALL_AXI	155	*STore Buffer* (STB) full with AXI requests causing CPU to stall
0xC50B	STB_FULL_STALL_TCM	156	STB full with TCM requests causing CPU to stall
0xC50C	CPU_STALLED_AHBS	157	CPU is stalled because TCM access through AHBS
0xC50D	AHBS_STALLED_CPU	158	AHBS is stalled due to TCM access by CPU
0xC50E	BR_INTERSTATING_MIS_PRED	159	Inter-stating branch is mis-predicted.
0xC50F	DWT_STALL	160	*Data Watchpoint and Trace* (DWT) stall
0xC510	DWT_FLUSH	161	DWT flush
0xC511	ETM_STALL	162	*Embedded Trace Macrocell* (ETM) stall
0xC512	ETM_FLUSH	163	ETM flush
0xC513	ADDRESS_BANK_CONFLICT	164	Bank conflict prevents memory instruction dual issue
0xC514	BLOCKED_DUAL_ISSUE	165	Dual instruction issuing is prevented
0xC515	FP_CONTEXT_TRIGGER	166	Floating Point Context is created
0xC516	TAIL_CHAIN	167	New exception is handled without first unstacking
0xC517	LATE_ARRIVAL	168	Late-arriving exception taken during exception entry
0xC518	INT_STALL_FAULT	169	Delayed exception entry due to ongoing fault processing
0xC519	INT_STALL_DEV	170	Delayed exception entry due to outstanding device access
0xC51A	PAC_STALL	171	Stall caused by authentication code computation
0xC51B	PAC_RETIRED	172	PAC instruction architecturally executed
0xC51C	AUT_RETIRED	173	AUT instruction architecturally executed
0xC51D	BTI_RETIRED	174	BTI instruction architecturally executed
0xC51E	PF_NL_MODE	175	Prefetch in next line mode
0xC51F	PF_STREAM_MODE	176	Prefetch in stream mode
0xC520	PF_BUFF_CACHE_HIT	177	Prefetch request that hit in the cache
0xC521	PF_REQ_LFB_HIT	178	Prefetch request that hit in line fill buffers
0xC522	PF_BUFF_FULL	179	Number of times prefetch buffer is full
0xC523	PF_REQ_DCACHE_HIT	180	Generated prefetch request address that hit in D-Cache

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

- PMU event numbers 0-120 are architectural, and 121-140 are Cortex®-M85-specific.
- All events are exported to the external output signal **EVENTBUS** as a single cycle pulse allowing system level analysis of processor performance. In normal operation the **EVENTBUS** is only active when DWT, ITM, PMU or ETM trace is enabled. The **EVENTBUS** can be activated permanently by setting ACTLR.EVENTBUSEN.

16.3 PMU register summary

The following table shows the *Performance Monitoring Unit* (PMU) registers. Each of these registers are 32 bits wide.

For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.

Table 16-2: PMU register summary

Address	Name	Type	Reset value	Description
0xE0003000-0xE000301C	PMU_EVCNTR0-7	RW	0x0000XXXX	Performance Monitoring Unit Event Counter Register
0xE000307C	PMU_CCNTR	RW	UNKNOWN	Performance Monitoring Unit Cycle Counter Register
0xE0003400-0xE000341C	PMU_EVTYPER0-7	RW	0x0000XXXX	Performance Monitoring Unit Event Type and Filter Register
0xE000347C	PMU_CCFILTR	-	-	Reserved, **RES0**.
0xE0003C00	PMU_CNTENSET	RW	0x00000000	Performance Monitoring Unit Count Enable Set Register
0xE0003C20	PMU_CNTENCLR	RW	0x00000000	Performance Monitoring Unit Count Enable Clear Register
0xE0003C40	PMU_INTENSET	RW	0x00000000	Performance Monitoring Unit Interrupt Enable Set Register
0xE0003C60	PMU_INTENCLR	RW	0x00000000	Performance Monitoring Unit Interrupt Enable Clear Register
0xE0003C80	PMU_OVSCLR	RW	0x00000000	Performance Monitoring Unit Overflow Flag Status Clear Register
0xE0003CA0	PMU_SWINC	WO	0x00000000	Performance Monitoring Unit Software Increment Register
0xE0003CC0	PMU_OVSSET	RW	0x00000000	Performance Monitoring Unit Overflow Flag Status Set Register
0xE0003E00	PMU_TYPE	RO	0x00A05F08	Performance Monitoring Unit Type Register
0xE0003E04	PMU_CTRL	RW	0x00000XXX	Performance Monitoring Unit Control Register
0xE0003FB8	PMU_AUTHSTATUS	RO	0x00XX00XX	Performance Monitoring Unit Authentication Status Register
0xE0003FBC	PMU_DEVARCH	RO	0x47700A06	Performance Monitoring Unit Device Architecture Register
0xE0003FCC	PMU_DEVTYPE	RO	0x00000016	Performance Monitoring Unit Device Type Register
0xE0003FD0	PMU_PIDR4	RO	0x00000004	Performance Monitoring Unit Peripheral Identification Register 4
0xE0003FE0	PMU_PIDR0	RO	0x00000023	Performance Monitoring Unit Peripheral Identification Register 0
0xE0003FE4	PMU_PIDR1	RO	0x000000BD	Performance Monitoring Unit Peripheral Identification Register 1
0xE0003FE8	PMU_PIDR2	RO	0x0000000B	Performance Monitoring Unit Peripheral Identification Register 2
0xE0003FEC	PMU_PIDR3	RO	0x00000000	Performance Monitoring Unit Peripheral Identification Register 3
0xE0003FF0	PMU_CIDR0	RO	0x0000000D	Performance Monitoring Unit Component Identification Register 0

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE0003FF4	PMU_CIDR1	RO	0x00000090	Performance Monitoring Unit Component Identification Register 1
0xE0003FF8	PMU_CIDR2	RO	0x00000005	Performance Monitoring Unit Component Identification Register 2
0xE0003FFC	PMU_CIDR3	RO	0x000000B1	Performance Monitoring Unit Component Identification Register 3

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

17 Instrumentation Trace Macrocell

This chapter describes the *Instrumentation Trace Macrocell* (ITM).

17.1 ITM features

The Cortex®-M85 processor optionally implements the *Instrumentation Trace Macrocell* (ITM) which has the following features.

- Trace data generation. This includes:
    - `printf` style debugging using the stimulus port registers which generate instrumentation packets.
    - Global and local timestamp packet generation.
    - Synchronization packet generation.
- Arbitration between trace packets, that is, prioritizing multiple sources and selecting a single source at a time.
    - External *Data Watchpoint and Trace* (DWT) packets and internally generated packets.
    - This arbitration is done using a fixed priority scheme of the order:
        1. Synchronization requests.
        2. Stimulus.
        3. DWT.
        4. Local and global timestamps.
- Buffering packets in the FIFO before sending them to a trace sink over an AMBA® ATB interface, which is typically a CoreSight™ *Trace Port Interface Unit* (TPIU).
- Trace flush requests from the ATB interface.

The ITM functionality is predominantly architecturally defined. However, there are some **IMPLEMENTATION SPECIFIC** features.

For information on the architecturally-defined ITM functionality, see the *Arm®v8-M Architecture Reference Manual*.

The **IMPLEMENTATION SPECIFIC** information for the Cortex®-M85 ITM is detailed in this section.

Stimulus Ports

The ITM has 32 stimulus ports, the ITM_STIMn registers. This implies one ITM_TER register is included and ITM_TPR[31:4] is RAZ/WI. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.

The Security Extension does not require that any configuration registers are banked. The only requirement is that the trace is filtered appropriately. Therefore, the following apply.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Both Security states share the same stimulus and configuration registers.
- No trace messages are generated when non-invasive debug is disabled.
- Secure trace messages are only generated when secure non-invasive debug is enabled.

DWT packets

The ITM arbitrates the various packets that are generated before inserting them into the FIFO. The only exception to this are the global timestamps. *Data Watchpoint and Trace* (DWT) packets are taken one at a time in the order that DWT arbitration determines. A bus similar to an ATB bus is used between the DWT and ITM.

The DWT and ITM can generate ITM synchronization packets, global timestamps, and DSYNC pulses for synchronizing the trace stream. These are generated when ITM_TCR.SYNCENA is first enabled and then periodically generated using the DWT synchronization packet timer. For more information on the ITM_TCR registers, see the *Arm®v8-M Architecture Reference Manual*. The **DSYNC** pulse causes frame synchronization within the Cortex®-M85 *Trace Port Interface Unit* (TPIU) when connected to the **DSYNC** input on the unit. For more information on TPIU frame synchronization, see the *Arm® CoreSight™ Architecture Specification v3.0*.

It is also possible for a downstream CoreSight™ trace component to control when synchronization packets are generated by the ITM on ATB using the input **SYNCREQI** signal.

Local timestamp, LTS

The local timestamp counter is used to create a time delta between each LTS message.

Global timestamp, GTS

64-bit global timestamp packets can be generated from an external timer source.

Busy flag conditions

The ITM_TCR register includes BUSY status bit that indicates when the ITM is processing events, including all internally generated and DWT packets.

For more information on the ITM_TCR register, see *Arm®v8-M Architecture Reference Manual*.

Stimulus disabled bit

On read transactions, the ITM_STIMn.FIFOREADY indicates whether the local stimulus FIFO or buffer is ready to accept data. For more information on the ITM_STIMn register, see the *Arm®v8-M Architecture Reference Manual*.

Processor stalling for guaranteed trace

In some cases, the processor might need to be stalled to ensure that no trace data is lost because of FIFO overflow. This optional architectural feature can be enabled or disabled using the ITCM_TCR.STALLENA field. Using this feature might affect processor performance.

17.2 ITM register summary

The following table shows the *Instrumentation Trace Macrocell* (ITM) registers whose implementation is specific to this processor.

Other registers are described in the *Arm®v8-M Architecture Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Depending on the implementation of your processor, the ITM registers might not be present. Any register that is configured as not present reads as zero.

Note

- You must enable DEMCR.TRCENA before you program or use the ITM.
- If the ITM stream requires synchronization packets, you must configure the synchronization packet rate in the DWT.

Table 17-1: ITM register summary

Address	Name	Type	Reset	Description
0xE0000000- 0xE000007C	ITM_STIM0-ITM_STIM31	RW	0x00000002	ITM Stimulus Port Registers 0-31
0xE0000E00	ITM_TER	RW	0x00000000	ITM Trace Enable Register
0xE0000E40	ITM_TPR	RW	0x00000000	17.3 ITM_TPR, ITM Trace Privilege Register on page 254
0xE0000E80	ITM_TCR	RW	0x00000000	ITM Trace Control Register
0xE0000EF0	INT_ITREAD	RO	0x00000000	17.6 ITM_ITREAD, Integration Read Register on page 256
0xE0000EF8	INT_ITWRITE	WO	0x00000000	17.5 ITM_ITWRITE, Integration Write Register on page 255
0xE0000F00	ITM_ITCTRL	WO	0x00000000	17.4 ITM_ITCTRL, ITM Integration Mode Control Register on page 254
0xE0000FBC	ITM_DEVARCH	RO	0x47701A01	ITM CoreSight™ Device Architecture Register
0xE0000FCC	ITM_DEVTYPE	RW	0x00000043	ITM CoreSight™ Device Type Register
0xE0000FD0	ITM_PIDR4	RO	0x00000004	ITM Peripheral identification registers
0xE0000FD4	ITM_PIDR5	RO	0x00000000	
0xE0000FD8	ITM_PIDR6	RO	0x00000000	
0xE0000FDC	ITM_PIDR7	RO	0x00000000	
0xE0000FE0	ITM_PIDR0	RO	0x00000023	
0xE0000FE4	ITM_PIDR1	RO	0x000000BD	
0xE0000FE8	ITM_PIDR2	RO	0x0000000B	
0xE0000FEC	ITM_PIDR3	RO	0x00000000	
0xE0000FF0	ITM_CIDR0	RO	0x0000000D	ITM Component identification registers
0xE0000FF4	ITM_CIDR1	RO	0x00000090	
0xE0000FF8	ITM_CIDR2	RO	0x00000005	
0xE0000FFC	ITM_CIDR3	RO	0x000000B1	

Note

ITM registers are fully accessible in privileged mode.

In user mode:

- All registers can be read.
- Only the Stimulus registers and Trace Enable registers can be written, and only when the corresponding Trace Privilege Register bit is set.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Writes to registers other than the Stimulus registers and Trace Enable registers are invalid and they are ignored.

If Secure non-invasive debug authentication is not enabled in the Cortex®-M85 processor, writes to the Stimulus registers from the software running in Secure state are ignored.

17.3 ITM_TPR, ITM Trace Privilege Register

The ITM_TPR enables an operating system to control the stimulus ports that are accessible by user code.

Usage constraints

You can only write to this register in privileged mode.

Configurations

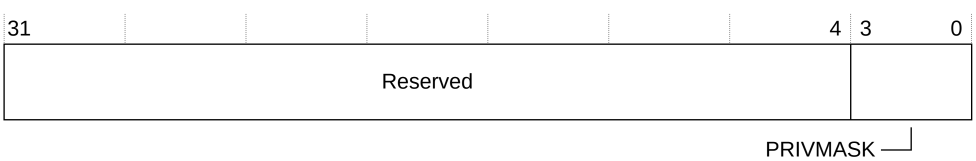
This register is available if the ITM is configured in your implementation.

Attributes

See 17.2 ITM register summary on page 252 for more information.

The following figure shows the ITM_TPR bit assignments.

Figure 17-1: ITM_TPR bit assignments



The following table shows the ITM_TPR bit assignments.

Table 17-2: ITM_TPR bit assignments

Bits	Name	Function
[31:4]	-	Reserved, RES0.
[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].

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

17.4 ITM_ITCTRL, ITM Integration Mode Control Register

The ITM_ITCTRL controls whether the trace unit is in integration mode.

Usage constraints

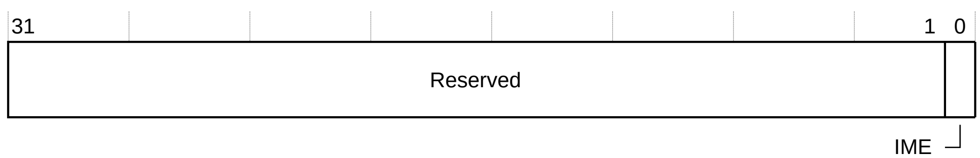
- Accessible from the memory-mapped interface or from an external agent such as a debugger.
- Arm recommends that you perform a debug reset after using integration mode. This register is write only and is only accessible in privilege mode.

Configurations Available in all configurations.

Attributes See 17.2 ITM register summary on page 252 for more information.

The following figure shows the ITM_ITCTRL bit assignments.

Figure 17-2: ITM_ITCTRL bit assignments



The following table shows the ITM_ITCTRL bit assignments.

Table 17-3: ITM_ITCTRL bit assignments

Bits	Name	Function
[31:1]	-	Reserved, **RES0**.
[0]	IME	Integration mode enable bit. The possible values are: **0** The trace unit is not in integration mode. **1** The trace unit is in integration mode. This mode enables: • A debug agent to perform topology detection. • SoC test software to perform integration testing.

17.5 ITM_ITWRITE, Integration Write Register

ITM_ITWRITE is used for integration testing.

Usage constraints This register is write only, and all reads are ignored. When ITM_ITCTRL.IME is not set and the processor is in privilege mode, then you can still write to this register. However, if the processor is not in privilege mode, then you cannot write to this register.

Configurations This register is:

- Only present in integration mode, when ITM_ITCTRL.IME is set to 1.

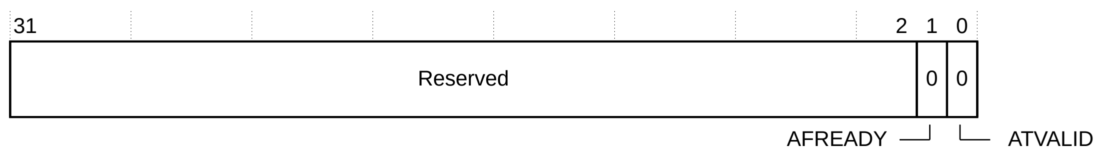
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Available in all configurations.

Attributes See 17.2 ITM register summary on page 252 for more information.

The following figure ITM_ITWRITE shows the bit assignments.

Figure 17-3: ITM_ITWRITE bit assignments



The following table shows the ITM_ITWRITE bit assignments.

Table 17-4: ITM_ITWRITE bit assignments

Bits	Name	Function
[31:2]	Reserved	RES0
[1]	AFREADY	When ITM_ITCTRL.IME is set, the value of this bit determines the value of **AFREADYI**. For more information on **AFREADYI**, see C.20 ITM interface signals on page 363 .
[0]	ATVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of **ATVALIDI**. For more information on **ATFVALIDI**, see C.20 ITM interface signals on page 363.

17.6 ITM_ITREAD, Integration Read Register

ITM_ITREAD is used for integration test.

Usage constraints This is a read-only register, and all writes are ignored. If ITM_ITCTRL.IME has not been set at all, then ITM_ITREAD.AFVALID and ITM_ITREAD_ATREADY bits return zero. However, in the case where ITM_ITCTRL.IME has been set at least once before, but is currently not set, then ITM_ITREAD.AFVALID and ITM_ITREAD.ATREADY return the previously stored **AFVALIDI** and **ATREADYI** values respectively.

Configurations This register is:

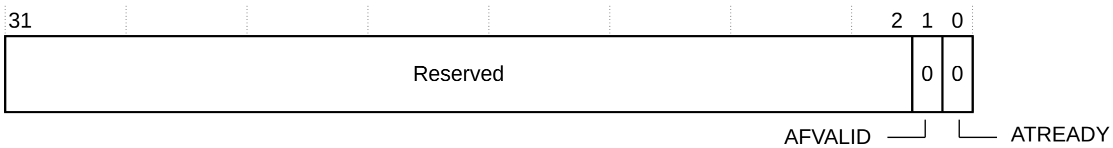
- Only present in integration mode, when ITM_ITCTRL.IME is set to 1.
- Available in all configurations.

Attributes See 17.2 ITM register summary on page 252 for more information.

The following figure ITM_ITREAD shows the bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 17-4: ITM_ITREAD bit assignments



The following table shows the ITM_ITREAD bit assignments.

Table 17-5: ITM_ITREAD bit assignments

Bits	Name	Function
[31:2]	Reserved	RES0
[1]	AFVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of **AFVALIDI**. When ITM_ITCTRL.IME is not set, this bit returns zero. For more information on **AFVALIDI**, see C.20 ITM interface signals on page 363.
[0]	ATREADY	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of **ATREADYI**. When ITM_ITCTRL.IME is not set, this bit returns zero. For more information on **ATREADYI**, see C.20 ITM interface signals on page 363.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

18 Data Watchpoint and Trace unit

This chapter describes the *Data Watchpoint and Trace* (DWT) unit.

18.1 DWT features

The Cortex®-M85 processor *Data Watchpoint and Trace* (DWT) unit has the following features:

- Watchpoints
- Data tracing
- Trace control signaling based on comparator match which can be used to control the *Cross Trigger Interface* (CTI) and optional *Embedded Trace Macrocell* (ETM) if it is configured in the processor
- *Program Counter* (PC) tracing
- Cycle count matching
- Additional PC sampling:
    - PC sample trace output as a result of a cycle count event
    - External PC sampling using a PC sample register
- Exception tracing
- Match event tracing
- Performance profiling counters
- An implementation of the *Performance Monitoring Unit* (PMU) that signals the DWT packet generator for PMU event overflow tracing
- Support for the *Digital Signal Processing* (DSP) extension

The DWT receives data transactions and instruction execution information from the processor core. Exception information and core profiling information is also delivered to the DWT from the processor core. The DWT comparators can be configured for two simultaneous data value comparisons.

The DWT compares instruction and data information using the comparators that are programmed according to the debug architecture. The results of these comparisons and any profiling counter and exception information are passed to the packet generator so it can generate, buffer, and arbitrate packets to be sent to the ITM.

Additional functionality includes ETM triggers using the **CMPMATCH** signals and invasive watchpoint debugging.

According to the architecture, all DWT debug events are asynchronous and are not recognized on the instruction which caused the event. Therefore the DWT PC-matching functionality cannot be used to implement breakpoints in the processor.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The Cortex®-M85 processor DWT supports tracing of exceptions using an interface to the processor. The exception state information is determined from the processor core exception control signals which indicate the following events:

- Idle.
- Exception entry.
- Exception exit.
- Exception return.

When exception trace is enabled in DWT_CTRL.EXCTRCENA, these events cause the DWT to output exception packets to the ITM.

Note

Data Trace Data Address packets are generated when there is a data address range match and if the comparator pair has been programmed accordingly. For more information on Data Trace Data Address packets, see the *Arm®v8-M Architecture Reference Manual*.

When there is a data address range match where the address of the first access is below the lower limit of the programmed address range, the Data Trace Data Address packet that is generated contains the address of the first access instead of the address of the first matching access. In this case, however, debugger tools can reconstruct the address of the first matching access by considering the following:

- A Data Trace Data Address packet has been generated, implying that there is a data address range match.
- The data address that is stored in the Data Trace Data Address packet is lower than the programmed lower range limit.

Therefore, the debugger tool can reconstruct the address of the first matching access to be equal to the programmed lower limit value of the address range.

18.2 DWT debug access control

The *Data Watchpoint and Trace* (DWT) features are dependent on whether DEMCR.TRCENA is set to enable trace and whether invasive or non-invasive debug is allowed at a given security level.

Invasive debug could possibly change the state of the processor. Non-invasive debug guarantees not to interfere or change the state of the processor. Both invasive and non-invasive debug provide memory access control, but there are certain restrictions on memory access control for non-invasive debug. For more information, see the *Arm®v8-M Architecture Reference Manual*.

The following table lists the DWT features for the possible invasive and non-invasive debug options.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 18-1: DWT debug access control

DEMCR.TRCENA	Invasive debug	Non-invasive debug	DWT features
0	Disabled	Disabled	No DWT watchpoints.
			Debugger accesses are blocked, except for CoreSight™ ID registers.
			Profiling and *Performance Monitoring Unit* (PMU) counters disabled. The DWT_CYCCNT (cycle counter) is disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
			DWT_PCSR reads `0xFFFFFFFF`.
	-	Enabled	No DWT watchpoints.
			Profiling and PMU counters disabled. The DWT_CYCCNT (cycle counter) is disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
			DWT_PCSR reads `0xFFFFFFFF`.
1	Disabled	Disabled	No DWT watchpoints.
			Debugger accesses are blocked, except for CoreSight™ ID registers.
			Profiling and PMU counters disabled. The DWT_CYCCNT (cycle counter) is not disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
			DWT_PCSR reads `0xFFFFFFFF`.
	Disabled	Enabled	No DWT watchpoints.
			Profiling and PMU counters enabled.
			Exception trace enabled.
			All comparators are enabled. This implies that there is data and instruction trace.
	Enabled	Enabled	Full DWT functionality.

Note

For a description of DEMCR and DWT_PCSR, see the *Arm®v8-M Architecture Reference Manual*.

18.3 DWT comparators

The *Data Watchpoint and Trace* (DWT) comparators offer various features which are adjusted based on the number of comparators supported in the Cortex®-M85 processor configuration.

The number of Data value matches depends on how many comparators are configured:

- A four comparator configuration supports one Data value match.
- An eight comparator configuration supports two simultaneous Data value matches.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The Arm debug architecture includes the facility to match on any address range by linking two comparators together, one marking the start of the range and the other marking the end of the range.

The following table shows the four comparator configuration, also referred to as the reduced set configuration.

Table 18-2: Four comparators configuration

`DBGLVL`	Comparator number	Instruction address matching	Data address matching	Cycle count matching	Data value matching	Supports linking?	Supports Watchpoint?	Supports CMPMATCH	Supports Trace?
1	0	Yes	Yes	Yes	No	No	Yes	Yes	Yes
	1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
	2	Yes	Yes	No	No	No	Yes	Yes	Yes
	3	Yes	Yes	No	No	Yes	Yes	Yes	Yes

The following table shows the eight comparator configuration, also referred to as the full set configuration.

Table 18-3: Eight comparators configuration

DBGLVL	Comparator number	Instruction address matching	Data address matching	Cycle count matching	Data value matching	Supports linking?	Supports Watchpoint?	Supports CMPMATCH	Supports Trace?
2	0	Yes	Yes	Yes	No	No	Yes	Yes	Yes
	1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
	2	Yes	Yes	No	No	No	Yes	Yes	Yes
	3	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
	4	Yes	Yes	No	No	No	Yes	Yes	No
	5	Yes	Yes	No	No	Yes	Yes	Yes	No
	6	Yes	Yes	No	No	No	Yes	Yes	No
	7	Yes	Yes	No	No	Yes	Yes	Yes	No

Note

- If linking is enabled on comparator 1, then there is no support for cycle count matching.
- For more information on determining the result of a comparator match that is done using the DWT_FUNCTION registers, see the *Arm®v8-M Architecture Reference Manual*.
- If the Cortex®-M85 processor is configured to include the *Embedded Trace Macrocell*, then the DWT can control trace start and stop functionality based on the comparator results using the CMPMATCH event, which is programmed using the DWT_FUNCTION registers.
- `DBGLVL` parameter determines whether four or eight DWT comparators are included. `DBGLVL`=0 is not supported.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

18.4 Cycle counter and profiling counters

The Cortex®-M85 DWT supports a cycle counter and profiling counters.

Cycle counter

When enabled in DWT_CTRL, the 32-bit cycle counter, DWT_CYCCNT, increments each cycle unless the processor is in debug halt state. When the cycle counter is disabled, all functionality associated with the cycle counter is also disabled.

The DWT_CTRL.CYCDISS bit field disables the cycle counter increment when the processor is executing secure code. This can be useful for generating CPI measurements for Non-secure applications.

Profiling counters

The profiling counters can be configured to generate events on overflow using DWT_CTRL fields.

CPI Counter (DWT_CPICNT)

The 8-bit CPI counter is incremented for every additional cycle, that is, greater than one taken to execute a non-load or store instruction. This counter must also be incremented for every cycle where fetch is stalled.

Exception Overhead Counter (DWT_EXCCNT)

The 8-bit Exception Overhead Counter is incremented for every cycle associated with exception entry and return. This includes stacking, unstacking, and preemption and tail-chaining, in cases where additional registers must be stacked due to a change in Security state between exceptions. Register stacking associated with floating-point lazy context saving is also included in this counter.

Sleep Overhead Counter (DWT_SLEEPCNT)

The 8-bit Sleep Overhead Counter is incremented for every cycle associated for power saving. For example, WFI and WFE exceptions.

Load-Store Counter (DWT_LSUCNT)

The 8-bit Load-Store Counter is incremented for every additional cycle that is greater than one taken to execute a load-store instruction.

Fold Counter (DWT_FOLDCNT)

The 8-bit Fold Counter counts folded instructions and increments for every instruction executed in zero cycles. All folded instructions are dual-issued. For example, for a dual-issued pair of instructions, the counter increments by one to reflect this.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

18.5 DWT register summary

The following table shows the *Data Watchpoint and Trace* (DWT) registers. Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero.

Table 18-4: DWT register summary

Address	Name	Type	Reset value	Description
0xE0001000	DWT_CTRL	RW	Possible reset values are: **0x48000000** Reduced DWT with no *Instrumentation Trace Macrocell* (ITM) trace **0x40000000** Reduced DWT with ITM trace **0x88000000** Full DWT with no ITM trace **0x80000000** Full DWT with ITM trace	DWT Control Register
0xE0001004	DWT_CYCCNT	RW	UNKNOWN	DWT Cycle Count Register
0xE0001008	DWT_CPICNT	RW	0x000000XX	DWT CPI Count Register
0xE000100C	DWT_EXCCNT	RW	0x000000XX	DWT Exception Overhead Count Register
0xE0001010	DWT_SLEEPCNT	RW	0x000000XX	DWT Sleep Count Register
0xE0001014	DWT_LSUCNT	RW	0x000000XX	DWT LSU Count Register
0xE0001018	DWT_FOLDCNT	RW	0x000000XX	DWT Folded-instruction Count Register
0xE000101C	DWT_PCSR	RO	UNKNOWN	DWT Program Counter Sample Register
0xE0001020	DWT_COMP0	RW	UNKNOWN	DWT Comparator Register 0
0xE0001028	DWT_FUNCTION0	RW	0x58000000	DWT Function Register 0
0xE0001030	DWT_COMP1	RW	UNKNOWN	DWT Comparator Register 1
0xE0001038	DWT_FUNCTION1	RW	0xF0000000	DWT Function Register 1
0xE0001040	DWT_COMP2	RW	UNKNOWN	DWT Comparator Register 2
0xE0001048	DWT_FUNCTION2	RW	0x50000000	DWT Function Register 2
0xE0001050	DWT_COMP3	RW	UNKNOWN	DWT Comparator Register 3

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE0001058	DWT_FUNCTION3	RW	Possible reset values are: **0xD0000000** Reduced DWT **0xF0000000** Full DWT	DWT Function Register 3
0xE0001060	DWT_COMP4	RW	UNKNOWN	DWT Comparator Register 4 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.
0xE0001068	DWT_FUNCTION4	RW	0x50000000	DWT Function Register 4
0xE0001070	DWT_COMP5	RW	UNKNOWN	DWT Comparator Register 5 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support. Can be linked to DWT_COMP4 to perform linked comparisons when DBGLVL=2.
0xE0001078	DWT_FUNCTION5	RW	0xD0000000	DWT Function Register 6
0xE0001080	DWT_COMP6	RW	UNKNOWN	DWT Comparator Register 6 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.
0xE0001088	DWT_FUNCTION6	RW	0x50000000	DWT Function Register 6
0xE0001090	DWT_COMP7	RW	UNKNOWN	DWT Comparator Register 7 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support. Can be linked to DWT_COMP6 to perform linked comparisons when DBGLVL=2.
0xE0001098	DWT_FUNCTION7	RW	0xD0000000	DWT Function Register 7
0xE000103C	DWT_VMASK1	RW	UNKNOWN	DWT Comparator Value Mask Register 0-14 DWT_VMASK3 is only present when DBGLVL=2. That is, when the processor is configured to have full set debug functionality, with eight DWT and eight BPU comparators. A maximum of two DWT_VMASK registers can be active. When DBGLVL=2, the comparators support two data value comparisons. Only comparators that can perform data value matching have corresponding DWT_VMASK registers. For more information on comparator configuration, see 18.3 DWT comparators on page 260
0xE000105C	DWT_VMASK3	RW		
0xE0001FBC	DWT_DEVARCH	RO	0x47711A02	DWT Device Type Architecture register
0xE0001FCC	DWT_DEVTYPE	RO	0x00000000	DWT Device Type Identifier register
0xE0001FD0	DWT_PIDR4	RO	0x00000004	DWT Peripheral identification registers 0-7
0xE0001FD4	DWT_PIDR5	RO	0x00000000	
0xE0001FD8	DWT_PIDR6	RO	0x00000000	
0xE0001FDC	DWT_PIDR7	RO	0x00000000	
0xE0001FE0	DWT_PIDR0	RO	0x00000023	
0xE0001FE4	DWT_PIDR1	RO	0x000000BD	
0xE0001FE8	DWT_PIDR2	RO	0x0000000B	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE0001FEC`	DWT_PIDR3	RO	`0x00000000`	
`0xE0001FF0`	DWT_CIDR0	RO	`0x0000000D`	DWT Component identification registers 0-3
`0xE0001FF4`	DWT_CIDR1	RO	`0x00000090`	
`0xE0001FF8`	DWT_CIDR2	RO	`0x00000005`	
`0xE0001FFC`	DWT_CIDR3	RO	`0x000000B1`	

DWT registers are described in the *Arm®v8-M Architecture Reference Manual*.

Note

- DWT_COMP4, DWT_COMP5, DWT_COMP6, and DWT_COMP7 can only be used for watchpoint and `CMPMATCH` and triggers and do not include data value or Trace support.
- DWT_COMP5 can be linked to DWT_COMP4 to perform linked comparisons when `DBGLVL`=2.
- DWT_COMP7 can be linked to DWT_COMP6 to perform linked comparisons when `DBGLVL`=2.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19 Cross Trigger Interface

This chapter describes the *Cross Trigger Interface* (CTI).

19.1 CTI features

The Cortex®-M85 processor *Cross Trigger Interface* (CTI) enables the processor debug logic and the *Embedded Trace Macrocell* (ETM) to interact with each other and with additional CoreSight™ debug and trace components in the system. This is done using trigger events across a standard interface and protocol. This allows software running on Cortex®-M85 to be debugged efficiently in both single processor systems and larger systems containing multiple processors.

The CTI is connected to a number of trigger inputs and outputs. The Cortex®-M85 CTI includes an external CTI channel interface with four input and four output channels. The input channel must be synchronous to **CLKIN**. The following figure shows the processor, ETM, CTI, and the available trigger input and output connections.

Note

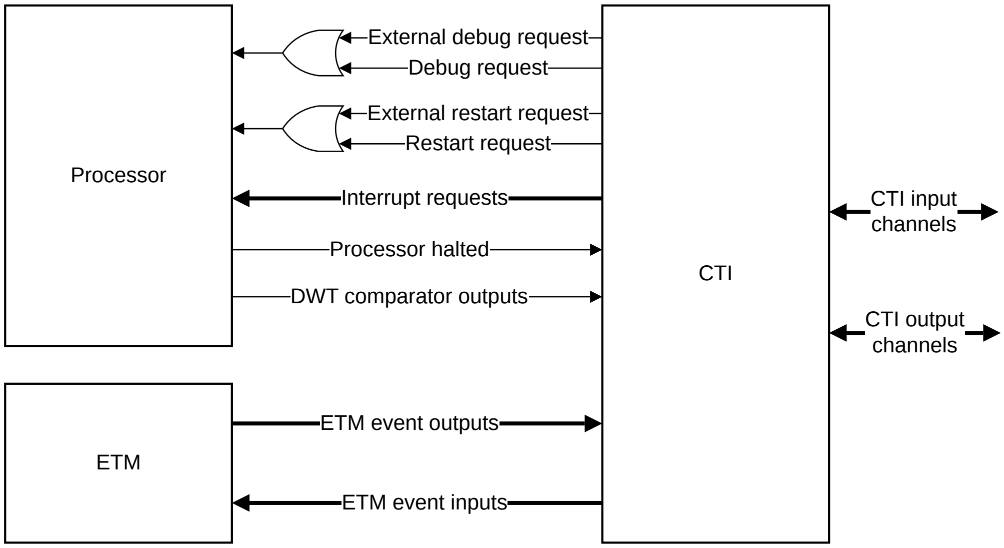
If the processor is configured with an ETM:

- Triggers 0-3 are connected to the event input and output signals.
- Up to a maximum of three *Data Watchpoint and Trace* (DWT) comparators (0, 1, and 2) can trigger events using **CMPMATCH**.

If the processor is not configured with an ETM, then the relevant triggers are not connected to the event input and output signals, and they are tied LOW.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 19-1: Cortex®-M85 processor CTI trigger connections



The following tables show the Cortex®-M85 processor CTI trigger signals assignment.

Table 19-1: Cortex®-M85 processor CTI input trigger signals assignment

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	Unused	ETM to CTI **Note:** If the ETM is not included, bits [4] and [5] are unused and tied LOW.	Pulsed
CTITRIGIN[6]	Unused		
CTITRIGIN[5]	ETM Event Output 1		
CTITRIGIN[4]	ETM Event Output 0 or DWT Comparator Output 3		
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	
CTITRIGIN[2]	DWT Comparator Output 1		
CTITRIGIN[1]	DWT Comparator Output 0		
CTITRIGIN[0]	Processor halted		

Table 19-2: Cortex®-M85 processor CTI output trigger signals assignment

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM **Note:** If the ETM is not included, bits[7:4] are unused and the output is left untied.	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		
CTITRIGOUT[5]	ETM Event Input 1		
CTITRIGOUT[4]	ETM Event Input 0		
CTITRIGOUT[3]	Interrupt Request 1	CTI to system	Acknowledged by software writing to CTI_INTACK register in the interrupt service routine.
CTITRIGOUT[2]	Interrupt Request 0		
CTITRIGOUT[1]	Processor Restart Request	CTI to processor	Processor restarted

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[0]	Processor Debug Halt Request		Acknowledged by the debugger writing to the CTI_INTACK register.

Note

The ETM is an optional licensable component. For more information on the ETM, see *Arm® CoreSight™ ETM-M85 Technical Reference Manual*.

19.2 CTI register summary

The following table shows the *Cross Trigger Interface* (CTI) programmable registers, with address offset, type, and reset value for each register.

Table 19-3: CTI register summary

Address	Name	Type	Reset value	Description
`0xE0042000`	CTI_CONTROL	RW	`0x00000000`	19.3 CTI_CONTROL, CTI Control Register on page 269
`0xE0042010`	CTI_INTACK	WO	`0x0000000X`	19.4 CTI_INTACK, CTI Interrupt Acknowledge Register on page 270
`0xE0042014`	CTI_APPSET	RW	`0x00000000`	19.5 CTI_APPSET, CTI Application Channel Set Register on page 271
`0xE0042018`	CTI_APPCLEAR	WO	`0x00000000`	19.6 CTI_APPCLR, CTI Application Channel Clear Register on page 272
`0xE004201C`	CTI_APPPULSE	WO	`0x00000000`	19.7 CTI_APPPULSE, CTI Application Channel Pulse Register on page 273
`0xE0042020`	CTI_INEN0	RW	`0x00000000`	19.8 CTI_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register on page 273
`0xE0042024`	CTI_INEN1	RW	`0x00000000`	
`0xE0042028`	CTI_INEN2	RW	`0x00000000`	
`0xE004202C`	CTI_INEN3	RW	`0x00000000`	
`0xE0042030`	CTI_INEN4	RW	`0x00000000`	
`0xE0042034`	CTI_INEN5	RW	`0x00000000`	
`0xE0042038`	CTI_INEN6	-	-	Reserved
`0xE004203C`	CTI_INEN7	-	-	
`0xE00420A0`	CTI_OUTEN0	RW	`0x00000000`	19.9 CTI_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register on page 275
`0xE00420A4`	CTI_OUTEN1	RW	`0x00000000`	
`0xE00420A8`	CTI_OUTEN2	RW	`0x00000000`	
`0xE00420AC`	CTI_OUTEN3	RW	`0x00000000`	
`0xE00420B0`	CTI_OUTEN4	RW	`0x00000000`	
`0xE00420B4`	CTI_OUTEN5	RW	`0x00000000`	
`0xE00420B8`	CTI_OUTEN6	RW	`0x00000000`	
`0xE00420BC`	CTI_OUTEN7	RW	`0x00000000`	
`0xE0042130`	CTI_TRIGINSTATUS	RO	UNKNOWN	19.10 CTI_TRIGINSTATUS, CTI Trigger Input Status Register on page 276
`0xE0042134`	CTI_TRIGOUTSTATUS	RO	UNKNOWN	19.11 CTI_TRIGOUTSTATUS, CTI Trigger Output Status Register on page 277

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
0xE0042138	CTI_CHINSTATUS	RO	0x0000000X	19.12 CTI_CHINSTATUS, CTI Channel Input Status Register on page 278
0xE004213C	CTI_CHOUTSTATUS	RO	0x0000000X	19.13 CTI_CHOUTSTATUS, CTI Channel Output Status Register on page 279
0xE0042140	CTI_CHANNELGATE	RW	0x0000000F	19.14 CTI_CHANNELGATE, CTI Channel Gate Register on page 279
0xE0042EE4	CTI_ITCHOUT	WO	0x00000000	19.15 CTI_ITCHOUT, Integration Test Channel Output Register on page 280
0xE0042EE8	CTI_ITTRIGOUT	WO	0x00000000	19.16 CTI_ITTRIGOUT, Integration Test Trigger Output Register on page 281
0xE0042EF4	CTI_ITCHIN	RO	0x00000000	19.17 CTI_ITCHIN, Integration Test Channel Input Register on page 283
0xE0042EF8	CTI_ITTRIGIN	RO	0x00000000	19.18 CTI_ITTRIGIN, Integration Test Trigger Input Register on page 283
0xE0042F00	CTI_ITCONTROL	RW	0x00000000	19.19 CTI_ITCONTROL, Integration Mode Control Register on page 285
0xE0042FBC	CTI_DEVARCH	RO	0x47701A14	19.20 CTI_DEVARCH, Device Architecture Register on page 285
0xE0042FC8	CTI_DEVID	RO	0x01040800	19.21 CTI_DEVID, Device Configuration Register on page 286
0xE0042FCC	CTI_DEVTYPE	RO	0x00000014	19.22 CTI_DEVTYPE, Device Type Identifier Register on page 287
0xE0042FD0	CTI_PIDR4	RO	0x00000004	19.23 CTI_PIDR4, Peripheral Identification Register 4 on page 288
0xE0042FD4	CTI_PIDR5	RO	0x00000000	19.24 CTI_PIDR5, Peripheral Identification Register 5 on page 289
0xE0042FD8	CTI_PIDR6	RO	0x00000000	19.25 CTI_PIDR6, Peripheral Identification Register 6 on page 289
0xE0042FDC	CTI_PIDR7	RO	0x00000000	19.26 CTI_PIDR7, Peripheral Identification Register 7 on page 290
0xE0042FE0	CTI_PIDR0	RO	0x00000023	19.27 CTI_PIDR0, Peripheral Identification Register 0 on page 291
0xE0042FE4	CTI_PIDR1	RO	0x000000BD	19.28 CTI_PIDR1, Peripheral Identification Register 1 on page 292
0xE0042FE8	CTI_PIDR2	RO	0x0000000B	19.29 CTI_PIDR2, Peripheral Identification Register 2 on page 292
0xE0042FEC	CTI_PIDR3	RO	0x00000000	19.30 CTI_PIDR3, Peripheral Identification Register 3 on page 293
0xE0042FF0	CTI_CIDR0	RO	0x0000000D	19.31 CTI_ CIDR0, Component Identification Register 0 on page 294
0xE0042FF4	CTI_CIDR1	RO	0x00000090	19.32 CTI_ CIDR1, Component Identification Register 1 on page 295
0xE0042FF8	CTI_CIDR2	RO	0x00000005	19.33 CTI_ CIDR2, Component Identification Register 2 on page 296
0xE0042FFC	CTI_CIDR3	RO	0x000000B1	19.34 CTI_ CIDR3, Component Identification Register 3 on page 296

19.3 CTI_CONTROL, CTI Control Register

The CTI_CONTROL register enables and disables the *Cross Trigger Interface* (CTI).

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented.

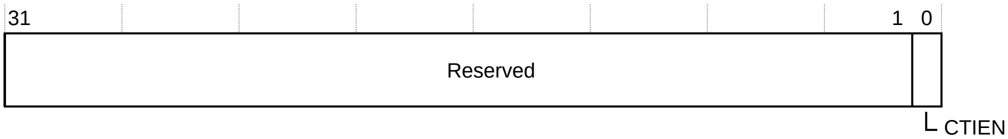
Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CONTROL bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 19-2: CTI_CONTROL bit assignments



The following table describes the CTI_CONTROL bit assignments.

Table 19-4: CTI_CONTROL bit assignments

Field	Name	Type	Description
[31:1]	Reserved	-	**RES0**
[0]	CTIEN	RW	Enable control. **0** CTI disabled. **1** CTI enabled. The reset value is `0b0`.

19.4 CTI_INTACK, CTI Interrupt Acknowledge Register

The CTI_INTACK register is a software acknowledge for trigger outputs. This register is a bit map that allows selective clearing of trigger output events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

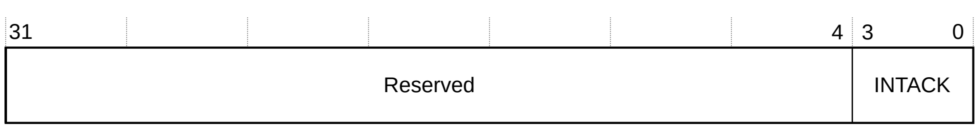
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_INTACK bit assignments.

Figure 19-3: CTI_INTACK bit assignments



The following table describes the CTI_INTACK bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table 19-5: CTI_INTACK bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	INTACK	WO	Acknowledges the corresponding **CTITRIGOUT[3:0]** output. **Note:** **INTACK[1]** is reserved. Writing 0x1 to this bit has no effect.

19.5 CTI_APPSET, CTI Application Channel Set Register

The CTI_APPSET register allows software to set any channel output. Software can use this register to generate a channel event in place of a hardware source on a trigger input.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

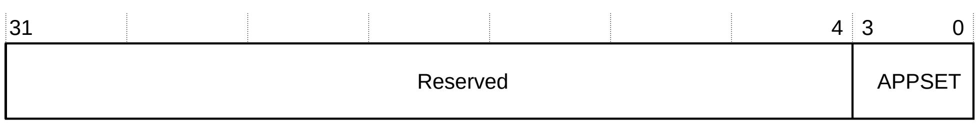
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_APPSET bit assignments.

Figure 19-4: CTI_APPSET bit assignments



The following table describes the CTI_APPSET bit assignments.

Table 19-6: CTI_APPSET bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	APPSET	RW	Sets the corresponding internal channel flag. **0** For reads, the application channel is inactive. For writes, this field has no effect. **1** For reads, the application channel is active. For writes, this field sets the channel output. The reset value is `0b0000`.

19.6 CTI_APPCLR, CTI Application Channel Clear Register

The CTI_APPCLR register allows software to clear any channel output. Software can use this register to clear a channel event instead of a hardware source on a trigger input.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

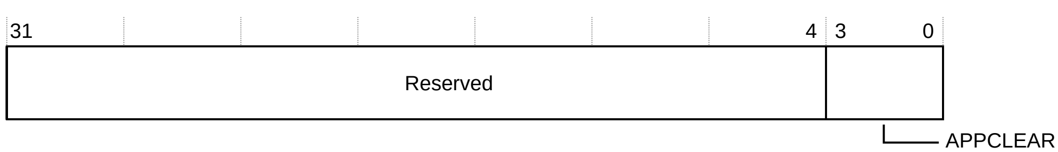
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_APPCLR bit assignments.

Figure 19-5: CTI_APPCLR bit assignments



The following table describes the CTI_APPCLR bit assignments.

Table 19-7: CTI_APPCLR bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	APPCLEAR	RW	Clears the corresponding internal channel flag. **0** This value has no effect. **1** This value clears the channel output. The reset value is `0b0000`.

19.7 CTI_APPPULSE, CTI Application Channel Pulse Register

The CTI_APPPULSE register allows software to pulse any channel output. Software can use this register to pulse a channel event in place of a hardware source on a trigger input.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

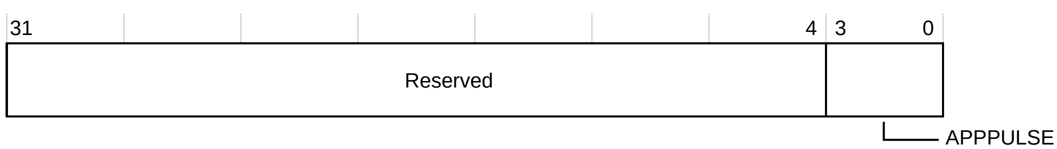
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_APPPULSE bit assignments.

Figure 19-6: CTI_APPPULSE bit assignments



The following table describes the CTI_APPPULSE bit assignments.

Table 19-8: CTI_APPPULSE bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**
[3:0]	APPPULSE	WO	Pulses the channel outputs. **0** This value has no effect. **1** Pulse channel event for one clock cycle.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.8 CTI_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register

The CTI_INEN<n> registers map trigger inputs to channels in the cross trigger system.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

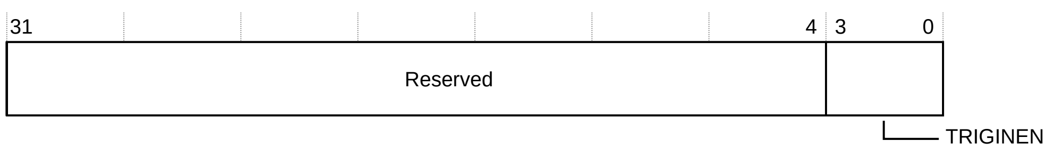
This register is always implemented.

Attributes

These are 32-bit registers. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_INEN<n> bit assignments, where n=0-5.

Figure 19-7: CTI_INEN<n> bit assignments, where n=0-5



The following table describes the CTI_INEN<n> bit assignments, where n=0-5.

Table 19-9: CTI_INEN<n> bit assignments, where n=0-5

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**
[3:0]	TRIGINEN	RW	Trigger input to channel mapping. **0** Input trigger events are ignored by the corresponding channel. **1** When an event is received on **CTITRIGIN**, an event is generated on the channel corresponding to this bit. The reset value is `0b0000`.

The following table provides more information on **CTITRIGIN** bit mapping.

Table 19-10: Cortex®-M85 processor CTI input trigger signals assignment

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	Unused	ETM to CTI **Note:** If the ETM is not included, bits [4] and [5] are unused and tied LOW.	Pulsed
CTITRIGIN[6]	Unused		
CTITRIGIN[5]	ETM Event Output 1		
CTITRIGIN[4]	ETM Event Output 0 or DWT Comparator Output 3		

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	
CTITRIGIN[2]	DWT Comparator Output 1		
CTITRIGIN[1]	DWT Comparator Output 0		
CTITRIGIN[0]	Processor halted		

19.9 CTI_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register

The CTI_OUTEN<n> registers map trigger outputs to channels in the cross trigger system.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

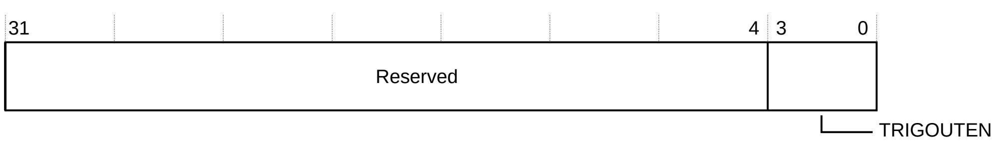
This register is always implemented.

Attributes

These are 32-bit registers. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_OUTEN<n> bit assignments, where n=0-7.

Figure 19-8: CTI_OUTEN<n> bit assignments, where n=0-7



The following table describes the CTI_OUTEN<n> bit assignments, where n=0-7.

Table 19-11: CTI_OUTEN<n> bit assignments, where n=0-7

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	TRIGOUTEN	RW	Channel to trigger enable mapping. **0** The corresponding channel is ignored by the output triggers. **1** When an event occurs on the channel corresponding to this bit, an event is generated on **CTITRIGOUT**. The reset value is `0b0000`.

The following table provides more information on **CTITRIGOUT** bit mapping.

Table 19-12: Cortex®-M85 processor CTI output trigger signals assignment

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM **Note:** If the ETM is not included, bits[7:4] are unused and the output is left untied.	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		
CTITRIGOUT[5]	ETM Event Input 1		
CTITRIGOUT[4]	ETM Event Input 0		
CTITRIGOUT[3]	Interrupt Request 1	CTI to system	Acknowledged by software writing to CTI_INTACK register in the interrupt service routine.
CTITRIGOUT[2]	Interrupt Request 0		
CTITRIGOUT[1]	Processor Restart Request	CTI to processor	Processor restarted
CTITRIGOUT[0]	Processor Debug Halt Request		Acknowledged by the debugger writing to the CTI_INTACK register.

19.10 CTI_TRIGINSTATUS, CTI Trigger Input Status Register

The CTI_TRIGINSTATUS register provides the trigger input status.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented.

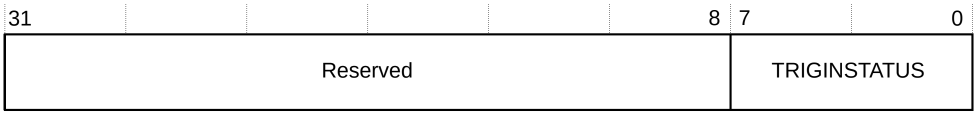
Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_TRIGINSTATUS bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 19-9: CTI_TRIGINSTATUS bit assignments



The following table describes the CTI_TRIGINSTATUS bit assignments.

Table 19-13: CTI_TRIGINSTATUS bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0
[7:0]	TRIGINSTATUS	RO	Trigger input status. One bit per trigger. **0** Input is LOW. **1** Input is HIGH. The reset value is UNKNOWN.

19.11 CTI_TRIGOUTSTATUS, CTI Trigger Output Status Register

The CTI_TRIGOUTSTATUS register provides the trigger output status.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

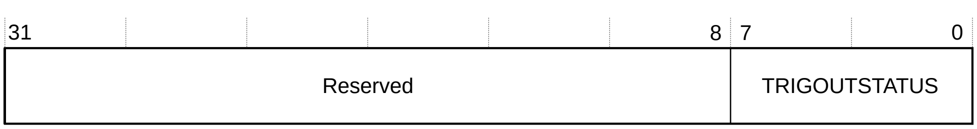
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_TRIGOUTSTATUS bit assignments.

Figure 19-10: CTI_TRIGOUTSTATUS bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the CTI_TRIGOUTSTATUS bit assignments.

Table 19-14: CTI_TRIGOUTSTATUS bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**
[7:0]	TRIGOUTSTATUS	RO	Trigger output status. One bit per trigger. **0** Output is LOW. **1** Output is HIGH. The reset value is **UNKNOWN**.

19.12 CTI_CHINSTATUS, CTI Channel Input Status Register

The CTI_CHINSTATUS register provides the channel input status.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

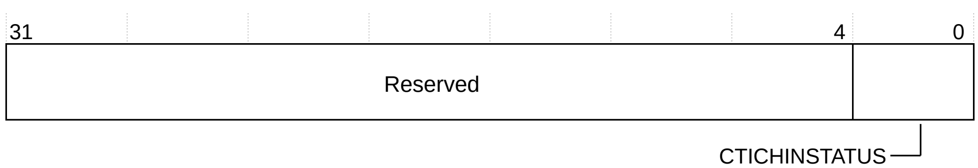
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CHINSTATUS bit assignments.

Figure 19-11: CTI_CHINSTATUS bit assignments



The following table describes the CTI_CHINSTATUS bit assignments.

Table 19-15: CTI_CHINSTATUS bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	CTICHINSTATUS	RO	Channel input status. One bit per channel input. **0** Input is LOW. **1** Input is HIGH. The reset value is **UNKNOWN**.

19.13 CTI_CHOUTSTATUS, CTI Channel Output Status Register

The CTI_CHOUTSTATUS register provides the channel output status.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

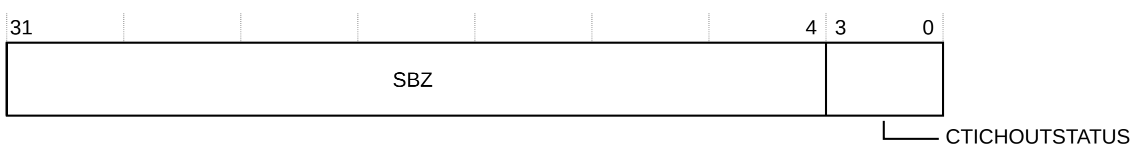
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CHOUTSTATUS bit assignments.

Figure 19-12: CTI_CHOUTSTATUS bit assignments



The following table describes the CTI_CHOUTSTATUS bit assignments.

Table 19-16: CTI_CHOUTSTATUS bit assignments

Field	Name	Type	Description
[31:4]	-	-	Reserved, **RES0**.
[3:0]	CTICHOUTSTATUS	RO	Channel output status. One bit per channel output. **0** Output is LOW. **1** Output is HIGH. The reset value is **UNKNOWN**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.14 CTI_CHANNELGATE, CTI Channel Gate Register

The CTI_CHANNELGATE register is the channel output gate.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

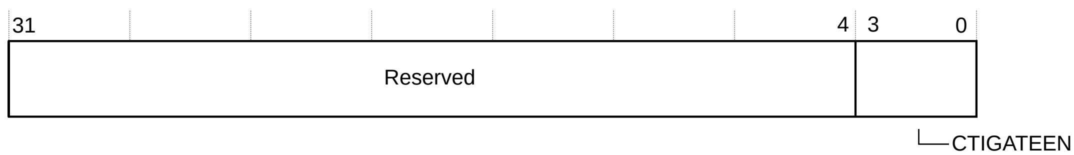
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CHANNELGATE bit assignments.

Figure 19-13: CTI_CHANNELGATE bit assignments



The following table describes the CTI_CHANNELGATE bit assignments.

Table 19-17: CTI_CHANNELGATE bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**
[3:0]	CTIGATEEN	RW	Enables the propagation of channel events out of the CTI. Propagation occurs one bit per channel. **0** Disable a channel from propagating. **1** Enable channel propagation. The reset value is `0b1111`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.15 CTI_ITCHOUT, Integration Test Channel Output Register

The CTI_ITCHOUT register is used to generate channel events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented.

Attributes

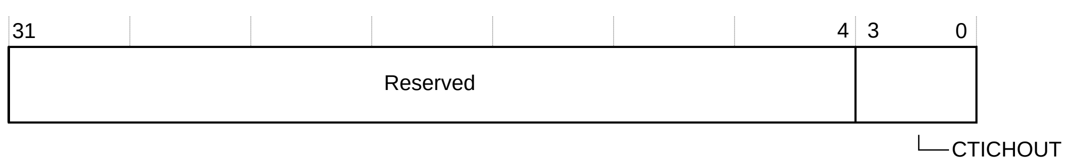
This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

Note

Writes to CTI_ITCHOUT and CTI_ITTRIGOUT only take effect when integration test mode is enabled using CTI_ITCONTROL.IME. For more information on CTI_ITCONTROL, see 19.19 CTI_ITCONTROL, Integration Mode Control Register on page 285.

The following figure shows the CTI_ITCHOUT bit assignments.

Figure 19-14: CTI_ITCHOUT bit assignments



The following table describes the CTI_ITCHOUT bit assignments.

Table 19-18: CTI_ITCHOUT bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CTICHOUT	WO	Pulses the channel outputs. **0** No effect. **1** Pulse channel event for one **CLKIN** cycle.

19.16 CTI_ITTRIGOUT, Integration Test Trigger Output Register

The CTI_ITTRIGOUT register is used to generate trigger events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

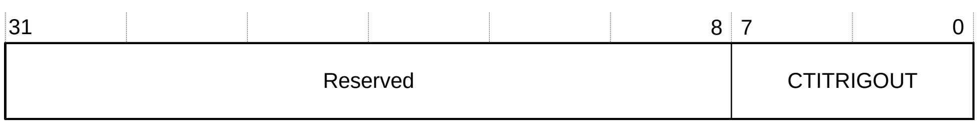
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_ITTRIGOUT bit assignments.

Figure 19-15: CTI_ITTRIGOUT bit assignments



The following table describes the CTI_ITTRIGOUT bit assignments.

Table 19-19: CTI_ITTRIGOUT bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**
[7:0]	CTITRIGOUT	WO	Set/clear trigger output signal. Some output triggers use a software handshake (CTITRIGOUT[3:0]), and others are pulsed (CTITRIGOUT[7:4]).

The following table provides more information on CTITRIGOUT bit mapping.

Table 19-20: Cortex®-M85 processor CTI output trigger signals assignment

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM **Note:** If the ETM is not included, bits[7:4] are unused and the output is left untied.	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		
CTITRIGOUT[5]	ETM Event Input 1		
CTITRIGOUT[4]	ETM Event Input 0		
CTITRIGOUT[3]	Interrupt Request 1	CTI to system	Acknowledged by software writing to CTI_INTACK register in the interrupt service routine.
CTITRIGOUT[2]	Interrupt Request 0		

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[1]	Processor Restart Request	CTI to processor	Processor restarted
CTITRIGOUT[0]	Processor Debug Halt Request		Acknowledged by the debugger writing to the CTI_INTACK register.

19.17 CTI_ITCHIN, Integration Test Channel Input Register

The CTI_ITCHIN register is used to view channel events. The integration test register includes a latch that is set when a pulse is received on a channel input. When read, a register bit reads as 1 if the channel has received a pulse since it was last read. The act of reading the register automatically clears the 1 to 0. When performing integration testing it is therefore important to coordinate the setting of event latches and reading/clearing them.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

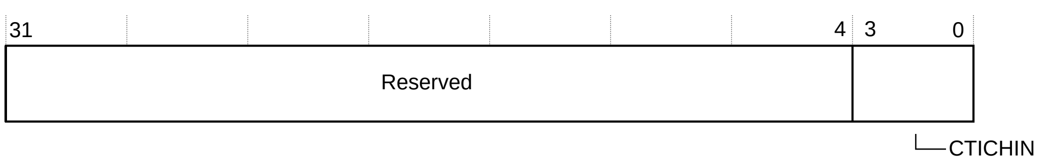
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_ITCHIN bit assignments.

Figure 19-16: CTI_ITCHIN bit assignments



The following table describes the CTI_ITCHIN bit assignments.

Table 19-21: CTI_ITCHIN bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**
[3:0]	CTICHIN	RO	Reads the latched value of the channel inputs. The reset value is `0b0000`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.18 CTI_ITTRIGIN, Integration Test Trigger Input Register

The CTI_ITTRIGIN register is used to view trigger events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

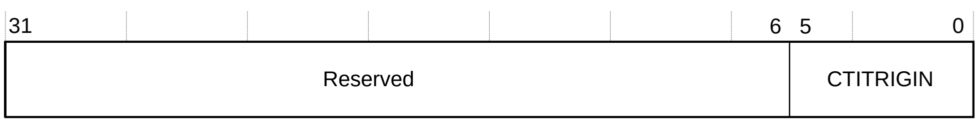
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_ITTRIGIN bit assignments.

Figure 19-17: CTI_ITTRIGIN bit assignments



The following table describes the CTI_ITTRIGIN bit assignments.

Table 19-22: CTI_ITTRIGIN bit assignments

Field	Name	Type	Description
[31:6]	Reserved	-	RES0.
[5:0]	CTITRIGIN	RO	Reads the latched value of the trigger inputs.

The following table provides more information on CTITRIGIN bit mapping.

Table 19-23: Cortex®-M85 processor CTI input trigger signals assignment

<table>
<tr><th>Signal</th><th>Description</th><th>Connection</th><th>Acknowledge, handshake</th></tr>
<tr><td>CTITRIGIN[7]</td><td>Unused</td><td rowspan="4">ETM to CTI
Note:
If the ETM is not included, bits [4] and [5] are unused and tied LOW.</td><td rowspan="8">Pulsed</td></tr>
<tr><td>CTITRIGIN[6]</td><td>Unused</td></tr>
<tr><td>CTITRIGIN[5]</td><td>ETM Event Output 1</td></tr>
<tr><td>CTITRIGIN[4]</td><td>ETM Event Output 0 or DWT Comparator Output 3</td></tr>
<tr><td>CTITRIGIN[3]</td><td>DWT Comparator Output 2</td><td rowspan="4">Processor to CTI</td></tr>
<tr><td>CTITRIGIN[2]</td><td>DWT Comparator Output 1</td></tr>
<tr><td>CTITRIGIN[1]</td><td>DWT Comparator Output 0</td></tr>
<tr><td>CTITRIGIN[0]</td><td>Processor halted</td></tr>
</table>

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.19 CTI_ITCONTROL, Integration Mode Control Register

The CTI_ITCONTROL register is used to enable topology detection.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

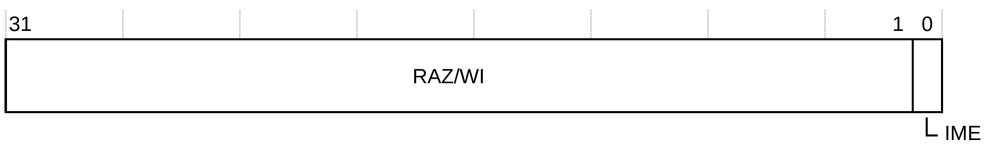
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_ITCONTROL bit assignments.

Figure 19-18: CTI_ITCONTROL bit assignments



The following table describes the CTI_ITCONTROL bit assignments.

Table 19-24: CTI_ITCONTROL bit assignments

Field	Name	Type	Description
[31:1]	RAZ/WI	-	Read-As-Zero, Writes Ignored.
[0]	IME	RW	Integration Mode Enable. When set, the component enters integration mode, enabling topology detection or integration testing to be performed. The reset value is `0b0`.

19.20 CTI_DEVARCH, Device Architecture Register

The CTI_DEVARCH register identifies the architect and architecture of the CoreSight™ *Cross Trigger Interface* (CTI).

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Configurations

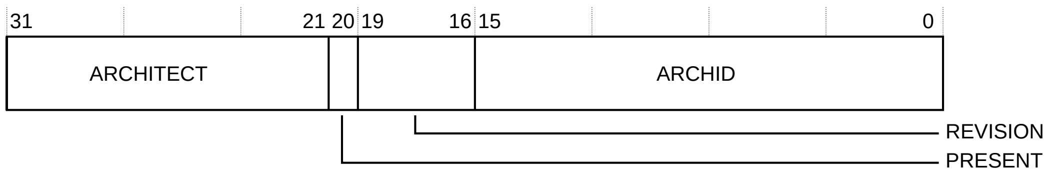
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_DEVARCH bit assignments.

Figure 19-19: CTI_DEVARCH bit assignments



The following table describes the CTI_DEVARCH bit assignments.

Table 19-25: CTI_DEVARCH bit assignments

Field	Name	Type	Description
[31:21]	ARCHITECT	RO	Defines the architect of the CTI. **[31:28]** Indicates the JEP106 continuation code. **[27:21]** Indicates the JEP106 identification code. Arm is the architect, therefore, this field is `0x23B`.
[20]	PRESENT	RO	Indicates the presence of this register. This field returns `0x1`.
[19:16]	REVISION	RO	Architecture revision. This field returns `0x0000`.
[15:0]	ARCHID	RO	Architecture ID. This field returns a value of `0x1A14`, indicating the CoreSight™ CTI architecture, version 3.0.

19.21 CTI_DEVID, Device Configuration Register

The CTI_DEVID register indicates the capability of the *Cross Trigger Interface* (CTI).

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

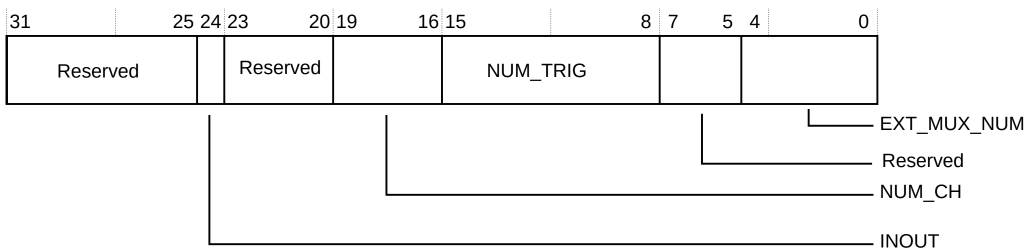
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_DEVID bit assignments.

Figure 19-20: CTI_DEVID bit assignments



The following table describes the CTI_DEVID bit assignments.

Table 19-26: CTI_DEVID bit assignments

Field	Name	Type	Description
[31:25]	Reserved	-	**RES0**.
[24]	INOUT	RO	Indicates that the CTIGATE register also masks the channel inputs. This field returns `0b0`. 19.14 CTI_CHANNELGATE, CTI Channel Gate Register on page 279.
[23:20]	Reserved	-	**RES0**.
[19:16]	NUM_CH	RO	The number of channels. This field returns `0b0100`.
[15:8]	NUM_TRIG	RO	Indicates the maximum number of triggers. This field returns `0b00001000`.
[7:5]	Reserved	-	**RES0**.
[4:0]	EXT_MUX_NUM	RO	This field is `0b00000` indicating that there is no multiplexing.

19.22 CTI_DEVTYPE, Device Type Identifier Register

A debugger can use the CTI_DEVTYPE register to get information about a component that has an unrecognized part number.

Usage constraints

This register is read-only.

Configurations

This register is always implemented.

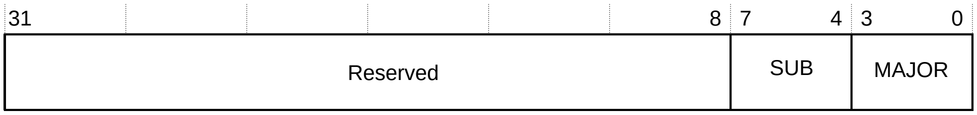
Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_DEVTYPE bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 19-21: CTI_DEVTYPE bit assignments



The following table describes the CTI_DEVTYPE bit assignments.

Table 19-27: CTI_DEVTYPE bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	SUB	RO	Minor classification. Returns `0x1`, indicating this component is a trigger matrix.
[3:0]	MAJOR	RO	Major classification. Returns `0x4`, indicating this component performs debug control.

19.23 CTI_PIDR4, Peripheral Identification Register 4

The CTI_PIDR4 register provides information about the memory size and JEP106 continuation code that the CoreSight™ *Cross Trigger Interface* (CTI) component uses.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

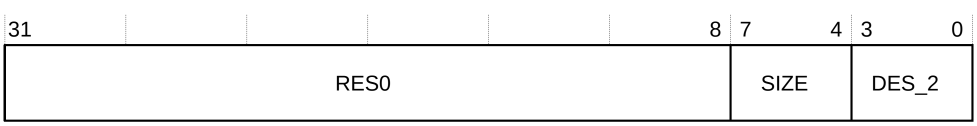
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR4 bit assignments.

Figure 19-22: CTI_PIDR4 bit assignments



The following table describes the CTI_PIDR4 bit assignments.

Table 19-28: CTI_PIDR4 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	SIZE	RO	This field indicates the memory size that the CTI uses. This field returns `0x0` indicating that the component uses an **UNKNOWN** number of 4KB blocks. The reset value of this field is `0x0`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	DES_2	RO	JEP106 continuation code. Together with CTI_PIDR2.DES_1 and CTI_PIDR1.DES_0, they indicate the designer of the component, not the implementer, except where the two are the same. The reset value of this field is `0x4`.

19.24 CTI_PIDR5, Peripheral Identification Register 5

The CTI_PIDR5 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

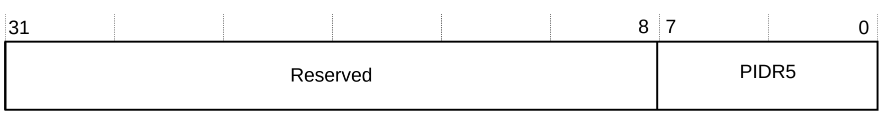
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR5 bit assignments.

Figure 19-23: CTI_PIDR5 bit assignments



The following table describes the CTI_PIDR5 bit assignments.

Table 19-29: CTI_PIDR5 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PIDR5	RO	**RES0**.

19.25 CTI_PIDR6, Peripheral Identification Register 6

The CTI_PIDR6 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Configurations

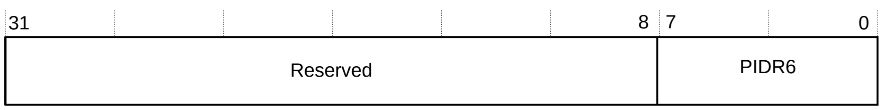
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR6 bit assignments.

Figure 19-24: CTI_PIDR6 bit assignments



The following table describes the CTI_PIDR6 bit assignments.

Table 19-30: CTI_PIDR6 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PIDR6	RO	**RES0**.

19.26 CTI_PIDR7, Peripheral Identification Register 7

The CTI_PIDR7 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented.

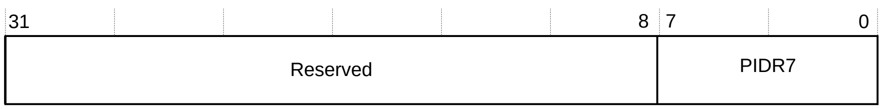
Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR7 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 19-25: CTI_PIDR7 bit assignments



The following table describes the CTI_PIDR7 bit assignments.

Table 19-31: CTI_PIDR7 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PIDR7	RO	**RES0**.

19.27 CTI_PIDR0, Peripheral Identification Register 0

The CTI_PIDR0 register indicates the *Cross Trigger Interface* (CTI) component part number.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

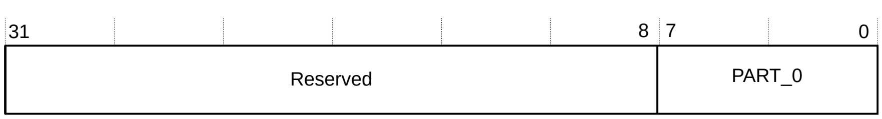
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR0 bit assignments.

Figure 19-26: CTI_PIDR0 bit assignments



The following table describes the CTI_PIDR0 bit assignments.

Table 19-32: CTI_PIDR0 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[7:0]	PART_0	RO	This field indicates the part number. When taken together with CTI_PIDR1.PART_1, it indicates the component. The part number is selected by the designer of the component. The reset value of this field is `0b00100011`.

19.28 CTI_PIDR1, Peripheral Identification Register 1

The CTI_PIDR1 register indicates the *Cross Trigger Interface* (CTI) component JEP106 continuation code and part number.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

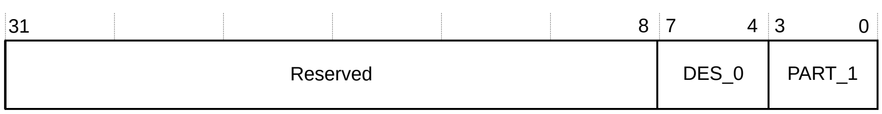
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR1 bit assignments.

Figure 19-27: CTI_PIDR1 bit assignments



The following table describes the CTI_PIDR1 bit assignments.

Table 19-33: CTI_PIDR1 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	DES_0	RO	This field indicates the JEP106 identification code, bits[3:0]. Together, with CTI_PIDR4.DES_2 and CTI_PIDR2.DES_1, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is `0xB`.
[3:0]	PART_1	RO	This field indicates the part number, bits[11:8]. Taken together with CTI_PIDR0.PART_0 it indicates the component. The part number is selected by the designer of the component. The reset value is `0xD`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.29 CTI_PIDR2, Peripheral Identification Register 2

The CTI_PIDR2 register indicates the *Cross Trigger Interface* (CTI) component revision number, JEDEC value, and part of the JEP106 continuation code.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

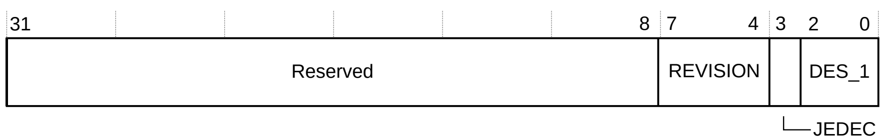
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR2 bit assignments.

Figure 19-28: CTI_PIDR2 bit assignments



The following table describes the CTI_PIDR2 bit assignments.

Table 19-34: CTI_PIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	REVISION	RO	This field indicates the revision number of the CTI component. It is an incremental value starting at `0x0` for the first design. The reset value is `0x0`.
[3]	JEDEC	RO	This field is always 1, indicating that a JEDEC assigned value is used.
[2:0]	DES_1	RO	This field is the JEP106 identification code, bits[6:4]. Together, with CTI_PIDR4.DES_2 and CTI_PIDR1.DES_0, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is `0b011`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.30 CTI_PIDR3, Peripheral Identification Register 3

The CTI_PIDR3 register indicates minor errata fixes of the *Cross Trigger Interface* (CTI) component and if you have modified the behavior of the component.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

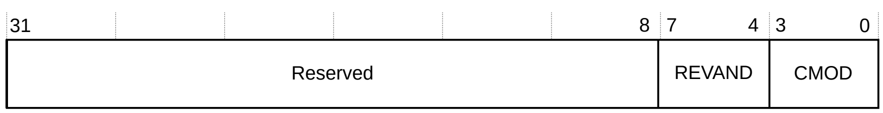
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_PIDR3 bit assignments.

Figure 19-29: CTI_PIDR3 bit assignments



The following table describes the CTI_PIDR3 bit assignments.

Table 19-35: CTI_PIDR3 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	REVAND	RO	This field indicates minor errata fixes specific to this design, for example metal fixes after implementation. This field is `0x0` without ECO.
[3:0]	CMOD	RO	Customer modified. Where the component is reusable IP, this value indicates whether you have modified the behavior of the component. This field is `0x0` without ECO.

19.31 CTI_ CIDR0, Component Identification Register 0

The CTI_CIDR0 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented.

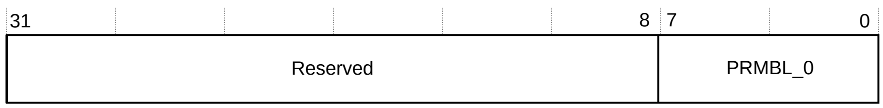
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CIDR0 bit assignments.

Figure 19-30: CTI_CIDR0 bit assignments



The following table describes the CTI_CIDR0 bit assignments.

Table 19-36: CTI_CIDR0 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PRMBL_0	RO	Preamble. This field returns `0x0D`.

19.32 CTI_ CIDR1, Component Identification Register 1

The CTI_CIDR1 register indicates the component class and preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

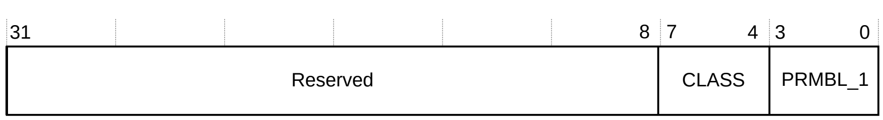
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CIDR1 bit assignments.

Figure 19-31: CTI_CIDR1 bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the CTI_CIDR1 bit assignments.

Table 19-37: CTI_CIDR1 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	CLASS	RO	Component class. Returns `0x9`, indicating this is a CoreSight™ component.
[3:0]	PRMBL_1	RO	Preamble. This field returns `0x0`.

19.33 CTI_ CIDR2, Component Identification Register 2

The CTI_CIDR2 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

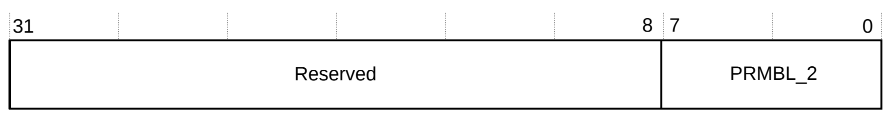
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CIDR2 bit assignments.

Figure 19-32: CTI_CIDR2 bit assignments



The following table describes the CTI_CIDR2 bit assignments.

Table 19-38: CTI_CIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PRMBL_2	RO	Preamble. This field returns `0x05`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

19.34 CTI_ CIDR3, Component Identification Register 3

The CTI_CIDR3 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

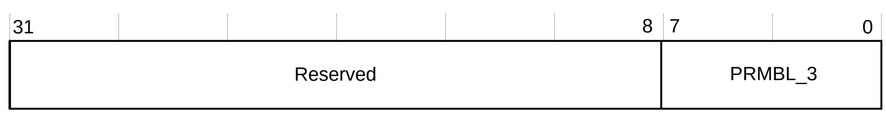
This register is always implemented.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the CTI_CIDR3 bit assignments.

Figure 19-33: CTI_CIDR3 bit assignments



The following table describes the CTI_CIDR3 bit assignments.

Table 19-39: CTI_CIDR3 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PRMBL_3	RO	Preamble. This field returns `0xB1`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

20 BreakPoint Unit

This chapter describes the *BreakPoint Unit* (BPU).

20.1 BPU features

The *BreakPoint Unit* (BPU) is an implementation of the architectural *Flash Patch and Breakpoint* (FPB) unit. The BPU can be configured with four or eight instruction address comparators. Each comparator supports breakpoint functionality on all instructions that are fetched across the entire address range in which code is located.

The BPU does not support flash patching. Flash patching allows a small programmable memory in the system to apply patches to program memory that cannot be modified.

The BPU functionality is largely architecturally defined. The **IMPLEMENTATION DEFINED** functionality includes:

Security

If invasive debug is not enabled for the security mode that the processor was in when the breakpoint became active, then debug events that are associated with breakpoints are blocked.

Architectural remap registers

The Cortex®-M85 processor does not include the address remapping functionality for instructions and literals. Therefore, the following architecturally defined registers have the following behavior:

- FP_REMAP.RMPSPT is RAZ/WI.
- FP_REMAP.REMAP is Reserved.
- FP_CTRL.NUM_LIT is 0, indicating that no literal comparators are included.
- Attempting to enable Flash Patch in FP_COMPn is ignored.

Also, only instruction address comparators are supported.

For more information on the registers listed in this section, see the *Arm®v8-M Architecture Reference Manual*.

20.2 BPU register summary

The following table shows the *BreakPoint Unit* (BPU) registers, with address, name, type and reset information for each register.

Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero and ignores writes.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

All BPU registers are described in the *Arm®v8-M Architecture Reference Manual*.

Table 20-1: BPU register summary

Address	Name	Type	Reset value	Description
0xE0002000	FP_CTRL	RW	• If four instruction comparators are implemented, the reset value is 0x10000040. • If eight instruction comparators are implemented, the rest value is 0x10000080.	Flash Patch Control Register
0xE0002004	FP_REMAP	RAZ/WI	-	Flash Patch Remap. This register is not implemented.
0xE0002008	FP_COMP0	RW	0x00000000	Flash Patch Comparator Register 0-7 **Note:** • FP_COMPn[0] is reset to 0. • FP_COMPn[31:1] is reset to **UNKNOWN**. • If only 4 breakpoints are implemented, FP_COMP4-FP_COMP7 are RAZ/WI.
0xE000200C	FP_COMP1	RW		
0xE0002010	FP_COMP2	RW		
0xE0002014	FP_COMP3	RW		
0xE0002018	FP_COMP4	RW		
0xE000201C	FP_COMP5	RW		
0xE0002020	FP_COMP6	RW		
0xE0002024	FP_COMP7	RW		
0xE0002FBC	FP_DEVARCH	RO	0x47701A03	FPB CoreSight™ Device Architecture Register
0xE0002FD0	FP_PIDR4	RO	0x00000004	Peripheral identification Register 4
0xE0002FE0	FP_PIDR0	RO	0x00000023	Peripheral identification Register 0
0xE0002FE4	FP_PIDR1	RO	0x000000BD	Peripheral identification Register 1
0xE0002FE8	FP_PIDR2	RO	0x0000000B	Peripheral identification Register 2
0xE0002FEC	FP_PIDR3	RO	0x00000000	Peripheral identification Register 3
0xE0002FF0	FP_CIDR0	RO	0x0000000D	Component identification registers
0xE0002FF4	FP_CIDR1	RO	0x00000090	
0xE0002FF8	FP_CIDR2	RO	0x00000005	
0xE0002FFC	FP_CIDR3	RO	0x000000B1	

Note

FP_DEVTYPE, FP_PIDR5, FP_PIDR6, and FP_PIDR7 registers are not implemented, and are **RES0**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Appendix A External Wakeup Interrupt Controller

This appendix describes the *External Wakeup Interrupt Controller* (EWIC) that can be used with the Cortex®-M85 processor.

A.1 EWIC features

The Cortex®-M85 processor supports the *External Wakeup Interrupt Controller* (EWIC), which is a peripheral to the processor and is suitable for sleep states when it is the only source of wakeup in the system. The EWIC stores state to allow the processor to wake up from retention or powered off state.

An APB interface controls the EWIC which must be connected to the *Core External Private Peripheral Bus* (Core EPPB) master interface of the processor. This interface is used to communicate all interrupt and event status information on sleep entry and wakeup. The EWIC interface can be asynchronous to the processor by instantiating an asynchronous clock domain crossing in the system on the APB interface.

EWIC configuration

The EWIC can be configured to support a variable number of events.

A minimum of 4 events are supported:

- External event
- Debug request
- Non-Maskable Interrupt, NMI
- One interrupt

A maximum of 483 events are supported:

- External event
- Debug request
- NMI
- 480 interrupts

Any number of events in the range 4-483 is permitted.

Note

The EWIC can support fewer interrupts than the processor supports. Interrupts above those that the EWIC supports cannot cause the core to exit low-power state. Therefore, higher numbered interrupts that occur when the core is in a low-power state might be lost.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

A.2 EWIC register summary

The *External Wakeup Interrupt Controller* (EWIC) requires memory-mapped registers that are accessed at address `0xE0047000` onwards in the PPB region of the memory map. The registers are contained in a CoreSight™ compliant 4KB block. The following table shows the EWIC registers.

Table A-1: EWIC register summary

Address	Name	Type	Reset value	Description
`0xE0047000`	EWIC_CR	RW	`0x00000000`	A.2.1 EWIC_CR, EWIC Control Register on page 301
`0xE0047004`	EWIC_ASCR	RW	`0x00000003`	A.2.2 EWIC_ASCR, EWIC Automatic Sequence Control Register on page 302
`0xE0047008`	EWIC_CLRMASK	WO	`0x00000000`	A.2.3 EWIC_CLRMASK, EWIC Clear Mask Register on page 304
`0xE004700C`	EWIC_NUMID	RO	`0x0000XXXX`	A.2.4 EWIC_NUMID, EWIC Event Number ID Register on page 304
`0xE0047200`	EWIC_MASKA	RW	`0x0000000X`	A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers on page 305
`0xE0047204` - `0xE004723C`	EWIC_MASKn	RW	UNKNOWN	
`0xE0047400`	EWIC_PENDA	RO	`0x0000000X`	A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers on page 306
`0xE0047404` - `0xE004743C`	EWIC_PENDn	RW	UNKNOWN	
`0xE0047600`	EWIC_PSR	RO	`0x0000XXXX`	A.2.7 EWIC_PSR, EWIC Pend Summary Register on page 308
`0xE0047604` -`0xE0047EFC`	-	UNK/ SBZP	-	Reserved
`0xE0047F00` -`0xE0047FFC`	CoreSight™ registers	RO	-	A.2.8 EWIC CoreSight register summary on page 309

A.2.1 EWIC_CR, EWIC Control Register

The EWIC_CR is the main *External Wakeup Interrupt Controller* (EWIC) control register.

Usage constraints

When the EWIC is connected to the *Core External Private Peripheral Bus* (Core EPPB) interface, the Cortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the EWIC is included.

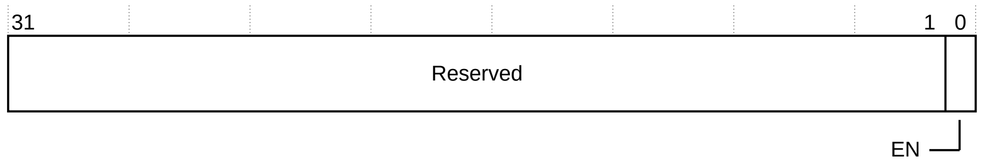
Attributes

This is a 32-bit register. See A.2 EWIC register summary on page 301 for more information.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following figure shows the EWIC_CR bit assignments.

Figure A-1: EWIC_CR bit assignments



The following table describes the EWIC_CR bit assignments.

Table A-2: EWIC_CR bit assignments

Field	Name	Type	Description
[31:1]	-	-	Reserved, **RES0**
[0]	EN	RW	The options are: **0** EWIC is disabled, events are not pended, and **WAKEUP** is not signaled. **1** EWIC is enabled, events are pended, and **WAKEUP** is signaled. The reset value is 0.

A.2.2 EWIC_ASCR, EWIC Automatic Sequence Control Register

The EWIC_ASCR determines whether the processor generates APB transactions on entry and exit from *Wakeup Interrupt Controller* (WIC) sleep to set up the wakeup state in the *External Wakeup Interrupt Controller* (EWIC).

Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the EWIC is included.

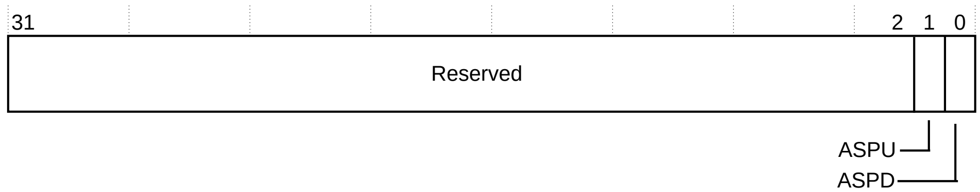
Attributes

This is a 32-bit register. See A.2 EWIC register summary on page 301 for more information.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following figure shows the EWIC_ASCR bit assignments.

Figure A-2: EWIC_ASCR bit assignments



The following table describes the EWIC_ASCR bit assignments.

Table A-3: EWIC_ASCR bit assignments

Field	Name	Type	Description
[31:2]	-	-	Reserved, **RES0**
[1]	ASPU	RW	The value of this bit is sent to the processor. The processor must use this value to decide whether any automatic EWIC accesses must be performed on transitioning from a low-power state. The options are: **0** No automatic sequence on powerup. **1** Automatic sequence on powerup. The reset value is 1.
[0]	ASPD	RW	The value of this bit is sent to the processor. The processor must use this value to decide whether any automatic EWIC accesses must be performed on transitioning to a low-power state. The options are: **0** No automatic sequence on entry to a low-power state. **1** Automatic sequence on entry to a low-power state. The reset value is 1.

Note

- If the automatic sequence is disabled, then software can program the unit by writing to the EWIC_MASKA and EWIC_MASKn registers on sleep entry and reading from the EWIC_PENDn registers on sleep exit. For more information, see A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers on page

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

305 and A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers on page 306.

- The value of EWIC_ASCR does not affect the operation of the EWIC itself. It only affects the control information that is driven on the **WICCONTROL** signal to the Cortex®-M85 processor.
- When modifying EWIC_ASCR.ASPU and EWIC_ACSR.ASPD, the resulting changes to **WICCONTROL[3:0]** must be stable before software enters sleep and remain stable until software execution resumes. Otherwise, modification of these registers can result in **UNPREDICTABLE** behavior.

A.2.3 EWIC_CLRMASK, EWIC Clear Mask Register

When there is a write to the EWIC_CLRMASK register, it causes EWIC_MASKA and all the EWIC_MASKn registers to be cleared. The write data is ignored. This register is RAZ.

A.2.4 EWIC_NUMID, EWIC Event Number ID Register

The EWIC_NUMID register returns the total number of events that are supported in the *External Wakeup Interrupt Controller* (EWIC).

Usage constraints

When the EWIC is connected to the *Core External Private Peripheral Bus* (Core EPPB) interface, the Cortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

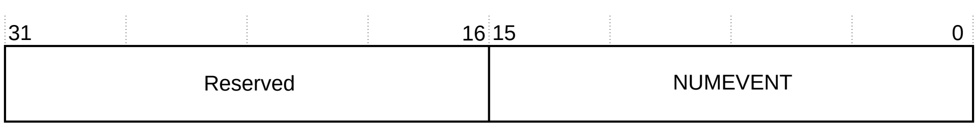
This register is always implemented when the EWIC is included.

Attributes

This is a 32-bit register. See A.2 EWIC register summary on page 301 for more information.

The following figure shows the EWIC_NUMID bit assignments.

Figure A-3: EWIC_NUMID bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the EWIC_NUMID bit assignments.

Table A-4: EWIC_NUMID bit assignments

Field	Name	Type	Description
[31:16]	-	-	Reserved, RES0
[15:0]	NUMEVENT	RO	The number of events supported.

A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers

The EWIC_MASKA register defines the mask for special events and the EWIC_MASKn registers for external interrupt (IRQ) events. There is one EWIC_MASKn register implemented for every 32 external interrupts that the *External Wakeup Interrupt Controller* (EWIC) supports. At least one register is always implemented. EWIC_MASKn is at address `0xE0047204`+(n×4), where n=0-14.

Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

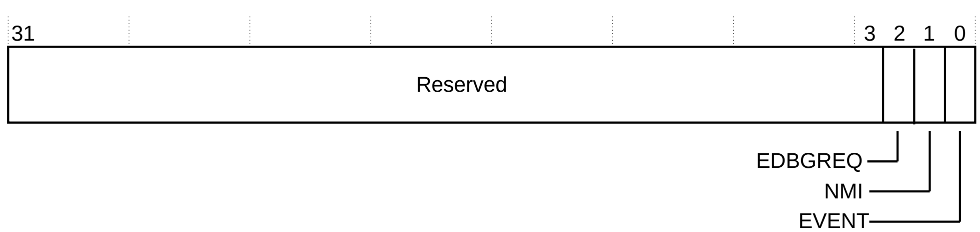
These registers are always implemented when the EWIC is included.

Attributes

These are 32-bit registers. See A.2 EWIC register summary on page 301 for more information.

The following figure shows the EWIC_MASKA bit assignments.

Figure A-4: EWIC_MASKA bit assignments



The following table describes the EWIC_MASKA bit assignments.

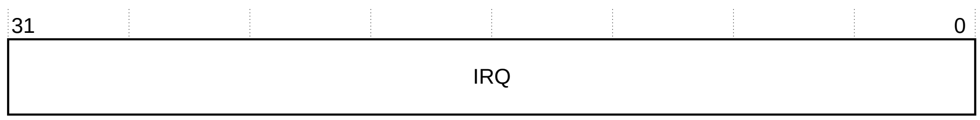
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table A-5: EWIC_MASKA bit assignments

Field	Name	Type	Description
[31:3]	-	-	Reserved, RES0
[2]	EDBGREQ	RW	Mask for external debug request. If this bit is 0, the mask is enabled.
[1]	NMI	RW	Mask for Non-Maskable Interrupt, NMI. If this bit is 0, the mask is enabled.
[0]	EVENT	RW	Mask for *Wait For Exception* (WFE) wakeup event. If this bit is 0, the mask is enabled.

The following figure shows the EWIC_MASKn, where n=0-14, bit assignments.

Figure A-5: EWIC_MASKn, where n=0-14 bit assignments



The following table describes the EWIC_MASKn, where n=0-14, bit assignments.

Table A-6: EWIC_MASKn, where n=0-14, bit assignments

Field	Name	Type	Description
[31:0]	IRQ	RW	Masks for external interrupts (n×32) to ((n+1)×32)-1. If any of the bits are 0, the mask is enabled for the associated interrupt. Additionally, any interrupt that the WIC does not support is also RAZ.

A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers

These registers indicate which events have been pended. The EWIC_PENDA register is used for special events and the EWIC_PENDn registers are used for external interrupt (IRQ) events. There is one EWIC_PENDn register implemented for each 32 external interrupt events the EWIC supports. EWIC_PENDA and at least one EWIC_PENDn register is always implemented.

Usage constraints

When the EWIC is connected to the *Core External Private Peripheral Bus* (Core EPPB) interface, the Cortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

These registers are always implemented when the EWIC is included. There is one EWIC_PENDn register implemented for every 32 events that the *External Wakeup Interrupt Controller* (EWIC) supports. At least one register is always implemented. EWIC_MASKn is at address `0xE0047404`+(n×4).

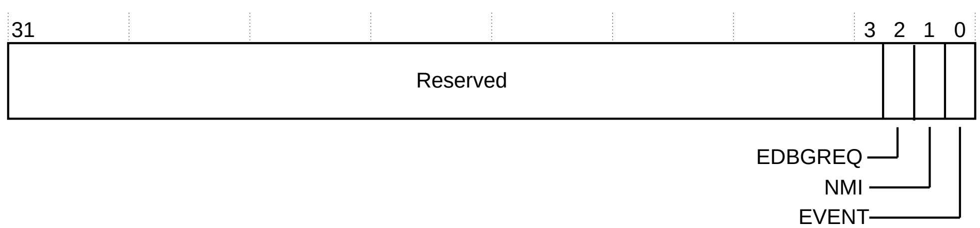
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

These are 32-bit registers. The EWIC_PENDn registers can be written to transfer pended interrupts in the NVIC when the processor enters sleep. EWIC_PENDA is read-only as special events can only be pended by the system (usually during sleep). See A.2 EWIC register summary on page 301 for more information.

The following figure shows the EWIC_PENDA bit assignments.

Figure A-6: EWIC_PENDA bit assignments



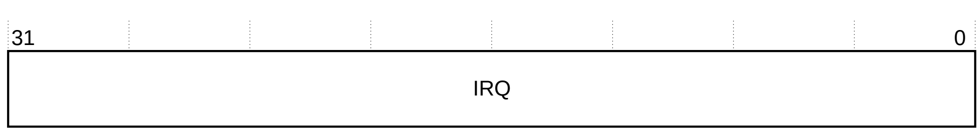
The following table describes the EWIC_PENDA bit assignments.

Table A-7: EWIC_PENDA bit assignments

Field	Name	Type	Description
[31:3]	-	-	Reserved, **RES0**
[2]	EDBGREQ	RO	External debug request is pended.
[1]	NMI	RO	Non-Maskable Interrupt, NMI, is pended.
[0]	EVENT	RO	*Wait For Exception* (WFE) wakeup event is pended.

The following figure shows the EWIC_PENDn, where n=0-14, bit assignments.

Figure A-7: EWIC_PENDn, where n=0-14 bit assignments



The following table describes the EWIC_PENDn, where n=0-14, bit assignments.

Table A-8: EWIC_PENDn, where n=0-14, bit assignments

Field	Name	Type	Description
[31:0]	IRQ	RW	Interrupts (n×32) to ((n+1)×32)-1 are pended. A write of zero to this field is ignored.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

Any IRQ bit associated with an interrupt that the EWIC does not support is RAZ/WI. All EWIC_PENDn registers are reset 0. If an event occurs when EWIC_CR.EN is set, then the corresponding bit in EWIC_PENDn is set. All EWIC_PENDn registers are cleared if the EWIC is disabled, that is, if EWIC_CR.EN is cleared. For more information on EWIC_CR, see A.2.1 EWIC_CR, EWIC Control Register on page 301.

A.2.7 EWIC_PSR, EWIC Pend Summary Register

The EWIC_PSR indicates which EWIC_PENDn registers are nonzero. This allows the processor to efficiently determine which EWIC_PENDn registers need to be read. This can be used to improve code efficiency in the powerup sequence.

Usage constraints

When the EWIC is connected to the *Core External Private Peripheral Bus* (Core EPPB) interface, theCortex®-M85 processor controls access to these registers using the following constraints:

- Access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

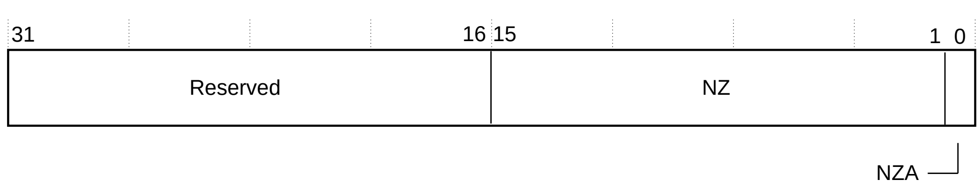
This register is always implemented when the EWIC is included.

Attributes

This is a 32-bit register. See A.2 EWIC register summary on page 301 for more information.

The following figure shows the EWIC_PSR bit assignments.

Figure A-8: EWIC_PSR bit assignments



The following table describes the EWIC_PSR bit assignments.

Table A-9: EWIC_PSR bit assignments

Field	Name	Type	Description
[31:16]	-	-	Reserved, **RES0**
[15:1]	NZ	RO	If EWIC_PSR.NZ[n+1] is set, then EWIC_PENDn is nonzero.

Field	Name	Type	Description
[0]	NZA	RO	If EWIC_PSR.NZA set, then EWIC_PENDA is nonzero.

Note

If any bit of EWIC_PSR is associated with an EWIC_PENDn register that is entirely RAZ/WI, then the bit in EWIC_PSR is also RAZ/WI.

A.2.8 EWIC CoreSight™ register summary

The *External Wakeup Interrupt Controller* (EWIC) implements the standard CoreSight™ registers.

The following table describes the CoreSight™ registers that the EWIC implements.

Table A-10: EWIC CoreSight™ register summary

Address	Name	Type	Reset value	Description
0xE0047F00	EWIC_ITCTRL	RO	0x00000000	Integration Mode Control Register
0xE0047F04-0xE0047F9C	-	-	-	Reserved
0xE0047FA0	EWIC_CLAIMSET	RW	0x0000000F	Claim Tag Set Register
0xE0047FA4	EWIC_CLAIMCLR	RW	0x00000000	Claim Tag Clear Register
0xE0047FA8	EWIC_DEVAFF0	RO	0x80000000	Device Affinity Register 0
0xE0047FAC	EWIC_DEVAFF1	RO	0x00000000	Device Affinity Register 1
0xE0047FB0	EWIC_LAR	WO	UNKNOWN	Lock Access Register
0xE0047FB4	EWIC_LSR	RO	0x00000000	Lock Status Register
0xE0047FB8	EWIC_AUTHSTATUS	RO	0x00000000	Authentication Status Register
0xE0047FBC	EWIC_DEVARCH	RO	0x47700A07	Device Architecture Register
0xE0047FC0	EWIC_DEVID2	RO	0x00000000	Device Configuration Register 2
0xE0047FC4	EWIC_DEVID1	RO	0x00000000	Device Configuration Register 1
0xE0047FC8	EWIC_DEVID	RO	0x00000000	Device Configuration Register
0xE0047FCC	EWIC_DEVTYPE	RO	0x00000000	Device Type Identifier Register
0xE0047FD0	EWIC_PIDR4	RO	0x00000004	Peripheral Identification Registers
0xE0047FD4	EWIC_PIDR5	RO	0x00000000	
0xE0047FD8	EWIC_PIDR6	RO	0x00000000	
0xE0047FDC	EWIC_PIDR7	RO	0x00000000	
0xE0047FE0	EWIC_PIDR0	RO	0x00000023	
0xE0047FE4	EWIC_PIDR1	RO	0x000000BD	
0xE0047FE8	EWIC_PIDR2	RO	0x0000000B	
0xE0047FEC	EWIC_PIDR3	RO	0x00000000	
0xE0047FF0	EWIC_CIDR0	RO	0x0000000D	Component Identification Registers
0xE0047FF4	EWIC_CIDR1	RO	0x00000090	
0xE0047FF8	EWIC_CIDR2	RO	0x00000005	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset value	Description
`0xE0047FFC`	EWIC_CIDR3	RO	`0x000000B1`	

Note

For more information on these registers, see the *Arm® CoreSight™ Architecture Specification v3.0*. In the *Arm® CoreSight™ Architecture Specification v3.0*, these register names are not prefixed with "EWIC_".

A.2.9 EWIC_CLAIMSET, EWIC Claim Tag Set Register

The EWIC_CLAIMSET register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

Usage constraints

See A.2.8 EWIC CoreSight register summary on page 309 for more information.

Configurations

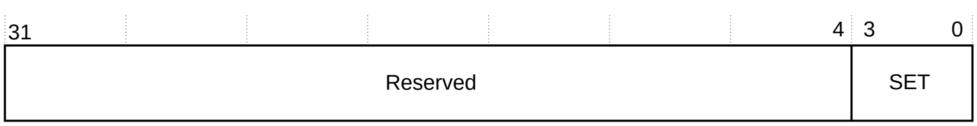
This register is always implemented.

Attributes

This is a 32-bit register.

The following figure shows the EWIC_CLAIMSET bit assignments.

Figure A-9: EWIC_CLAIMSET bit assignments



The following table describes the EWIC_CLAIMSET bit assignments.

Table A-11: EWIC_CLAIMSET bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	SET	RW	The options are: **Write 0** No effect. **Write 1** Set the claim tag for bit[n]. **Read 0** The claim tag that is represented by bit[n] is not implemented. **Read 1** The claim tag that is represented by bit[n] is implemented.

A.2.10 EWIC_CLAIMCLR, EWIC Claim Tag Clear Register

The EWIC_CLAIMCLR register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

Usage constraints

See A.2.8 EWIC CoreSight register summary on page 309 for more information.

Configurations

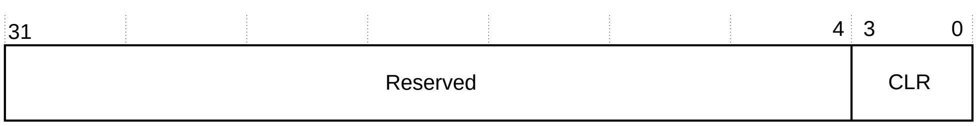
This register is always implemented.

Attributes

This is a 32-bit register.

The following figure shows the EWIC_CLAIMCLR bit assignments.

Figure A-10: EWIC_CLAIMCLR bit assignments



The following table describes the EWIC_CLAIMCLR bit assignments.

Table A-12: EWIC_CLAIMCLR bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	CLR	RW	The options are: **Write 0** No effect. **Write 1** Clear the claim tag for bit[n]. **Read 0** The claim tag that is represented by bit[n] is not set. **Read 1** The claim tag that is represented by bit[n] is set.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Appendix B Trace Port Interface Unit

This appendix describes the *Trace Port Interface Unit* (TPIU) that can be used with the Cortex®-M85 processor.

B.1 TPIU features

The Cortex®-M85 *Trace Port Interface Unit* (TPIU) is an optional component that bridges 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.

The Cortex®-M85 TPIU encapsulates IDs where required, and an external *Trace Port Analyzer* (TPA) captures the data stream.

The Cortex®-M85 TPIU is specially designed for low-cost debug. If your implementation requires additional debugging features, the following options are available:

- CoreSight™ TPIU-M, see the *Arm® CoreSight™ TPIU-M Technical Reference Manual* for more information
- CoreSight™ SoC-600 TPIU, see the *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* for more information

---

Note

In this chapter, the term TPIU refers to the Cortex®-M85 processor TPIU.

The *Trace Port Interface Unit* (TPIU) supports up to two ATB ports. The following table shows the various `ATB1` and `ATB2` parameters configuration options.

Table B-1: ATB port parameters

`ATB1`	`ATB2`	Description
0	0	Illegal combination. If the ITM and ETM do not exist, then the TPIU is not present.
0	1	ATB port 2 is present, and Arm recommends connecting the ETM to it. In this case, the ATB interface 2 logic is removed and gets assigned to ATB interface 1 logic.
1	0	ATB port 1 is present, and Arm recommends connecting the ITM to it.
1	1	Both ports are present, and Arm recommends that the ITM is connected to ATB port 1 and the ETM is connected to ATB port 2.

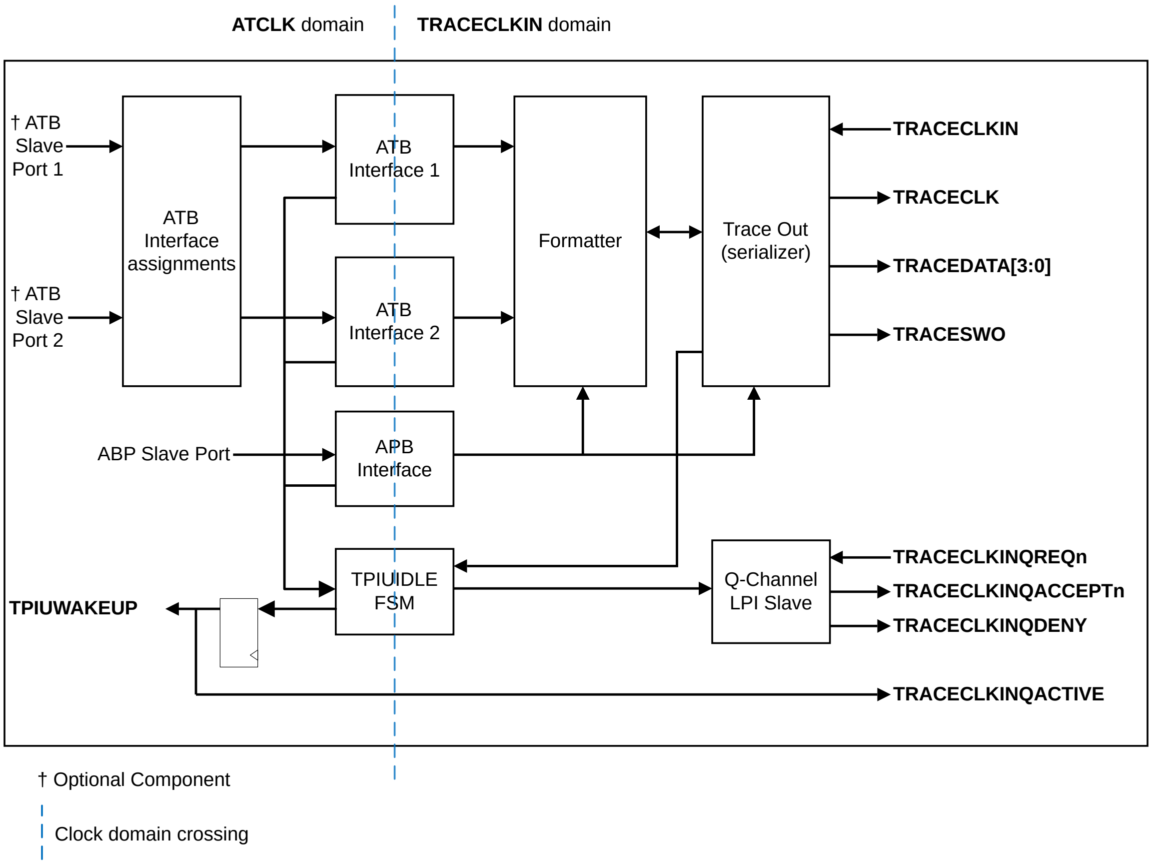
Note

If your system design uses the optional ETM component, the TPIU configuration supports both ITM and ETM debug trace. See the *Arm® CoreSight™ ETM-M85 Technical Reference Manual*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following figure shows the component layout of the TPIU when `ATB1` and `ATB2` are set to 1.

Figure B-1: TPIU block diagram



B.1.1 TPIU 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 Trace Port Mode is active.

The formatting protocol is described in the *Arm® CoreSight™ Architecture Specification v3.0*. You must enable synchronization in the DWT or TPIU_PSCR to provide synchronization for the formatter.

When the formatter is enabled, if there is no data to output after a frame has been started, half-sync packets can be inserted. Distributed synchronization from the DWT or TPIU_PSCR causes synchronization which ensures that any partial frame is completed, and at least one full synchronization packet is generated.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.1.2 Serial Wire Output format

The TPIU can output trace data in a *Serial Wire Output* (SWO) format:

- TPIU_DEVID specifies the formats that are supported. See B.2.16 TPIU_DEVID, Device Configuration Register on page 329
- TPIU_SPPR specifies the SWO format in use. See the *Arm®v8-M Architecture Reference Manual*.

When one of the two SWO modes is selected, you can enable the TPIU to bypass the formatter for trace output. When the formatter is bypassed, only data on the ATB interface 1 is passed through and ATB interface 2 data is discarded.

Note

When operating in bypass mode, Arm recommends that in a configuration that supports an ETM and ITM, the ITM data is passed through by connecting the ITM to the ATB Slave Port 1.

B.2 TPIU register summary

The following table shows the *Trace Port Interface Unit* (TPIU) registers. Depending on the implementation of your processor, the TPIU registers might not be present, and the CoreSight™ TPIU might be present instead. Any register that is configured as not present reads as zero.

Note

Arm recommends reprogramming the TPIU before any data has been presented on either ATB slave port and after either of the following:

- Both **ATRESETn** and **TRESETn** have been applied
- A flush has been completed using FFCR.FOnMan.

If this recommendation is not followed, reprogramming can cause either momentary or permanent data corruption that might require **ATRESETn** and **TRESETn** to be applied. This corruption is related to trace and not general data corruption of execution state or memory.

Table B-2: TPIU IMPLEMENTATION DEFINED register summary

Address	Name	Type	Reset	Description
`0xE0040000`	TPIU_SSPSR	RO	- **Note:** The value at reset corresponds to the **MAXPORTSIZE** configuration tie off.	B.2.1 TPIU_SSPSR, Supported Port Size Register on page 317
`0xE0040004`	TPIU_CSPSR	RW	`0x00000001`	B.2.2 TPIU_CSPSR, Current Port Size Register on page 318
`0xE0040010`	TPIU_ACPR	RW	`0x00000000`	B.2.5 TPIU_ACPR, Asynchronous Clock Prescaler Register on page 320

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset	Description
0xE00400F0	TPIU_SPPR	RW	0x00000001	B.2.3 TPIU_SPPR, Selected Pin Protocol Register on page 319
0xE0040300	TPIU_FFSR	RO	0x00000008	B.2.6 TPIU_FFSR, Formatter and Flush Status Register on page 321
0xE0040304	TPIU_FFCR	RW	0x00000102	B.2.7 TPIU_FFCR, Formatter and Flush Control Register on page 322
0xE0040308	TPIU_PSCR	RW	0x00000000	B.2.4 TPIU_PSCR, Periodic Synchronization Counter Register on page 319
0xE0040EE8	TPIU_TRIGGER	RO	0x00000000	B.2.8 TPIU_TRIGGER, TPIU TRIGGER Register on page 323
0xE0040EEC	TPIU_ITFTTD0	RO	UNKNOWN	B.2.9 ITFTTD0, Integration Test FIFO Test Data 0 Register on page 324
0xE0040EF0	TPIU_ITATBCTR2	RW	0x00000000	B.2.10 ITATBCTR2, Integration Test ATB Control Register 2 on page 325
0xE0040EF8	TPIU_ITATBCTR0	RO	0x00000000	B.2.12 ITATBCTR0, Integration Test ATB Control 0 Register on page 326
0xE0040EFC	TPIU_ ITFTTD1	RO	UNKNOWN	B.2.11 ITFTTD1, Integration Test FIFO Test Data 1 Register on page 325
0xE0040F00	TPIU_ITCTRL	RW	0x00000000	B.2.13 TPIU_ITCTRL, Integration Mode Control on page 327
0xE0040FA0	TPIU_CLAIMSET	RW	0x0000000F	B.2.14 CLAIMSET, Claim Tag Set Register on page 327
0xE0040FA4	TPIU_CLAIMCLR	RW	0x00000000	B.2.15 CLAIMCLR, Claim Tag Clear Register on page 328
0xE0040FC8	TPIU_DEVID	RO	0x00000CA0/0x00000CA1	B.2.16 TPIU_DEVID, Device Configuration Register on page 329
0xE0040FCC	TPIU_DEVTYPE	RO	0x00000011	B.2.17 TPIU_DEVTYPE, Device Type Identifier Register on page 331
0xE0040FD0	TPIU_PIDR4	RO	0x00000004	B.2.18 TPIU_PIDR4, Peripheral Identification Register 4 on page 331
0xE0040FD4	TPIU_PIDR5	RO	0x00000000	B.2.19 TPIU_PIDR5, Peripheral Identification Register 5 on page 332
0xE0040FD8	TPIU_PIDR6	RO	0x00000000	B.2.20 TPIU_PIDR6, Peripheral Identification Register 6 on page 333
0xE0040FDC	TPIU_PIDR7	RO	0x00000000	B.2.21 TPIU_PIDR7, Peripheral Identification Register 7 on page 334
0xE0040FE0	TPIU_PIDR0	RO	0x00000023	B.2.22 TPIU_PIDR0, Peripheral Identification Register 0 on page 334
0xE0040FE4	TPIU_PIDR1	RO	0x000000BD	B.2.23 TPIU_PIDR1, Peripheral Identification Register 1 on page 335
0xE0040FE8	TPIU_PIDR2	RO	0x0000000B	B.2.24 TPIU_PIDR2, Peripheral Identification Register 2 on page 336
0xE0040FEC	TPIU_PIDR3	RO	0x00000000 **Note:** The value of TPIU_PIDR3[7:4] is determined by MCU_ECOREVNUM[11:8].	B.2.25 TPIU_PIDR3, Peripheral Identification Register 3 on page 337

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Address	Name	Type	Reset	Description
0xE0040FF0	TPIU_CIDR0	RO	0x0000000D	B.2.26 TPIU_ CIDR0, Component Identification Register 0 on page 338
0xE0040FF4	TPIU_CIDR1	RO	0x00000090	B.2.27 TPIU_ CIDR1, Component Identification Register 1 on page 338
0xE0040FF8	TPIU_CIDR2	RO	0x00000005	B.2.28 TPIU_ CIDR2, Component Identification Register 2 on page 339
0xE0040FFC	TPIU_CIDR3	RO	0x000000B1	B.2.29 TPIU_ CIDR3, Component Identification Register 3 on page 340

B.2.1 TPIU_SSPSR, Supported Port Size Register

TPIU_SSPSR shows the supported sizes of the trace data port **TRACEDATE[3:0]**. Each bit location represents a single port size that is supported, that is, sizes from 32 bits to 1 bit in bit location [31:0]. If a bit is set, then that port size is supported. The supported trace port sizes are limited by the MAXPORTSIZE signal. The maximum possible trace port size for Cortex®-M85 is 4 bits.

For more information on the **MAXPORTSIZE** signal, see the *Arm® Cortex®-M85 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M85 Processor Integration and Implementation Manual* is a confidential document and available to licensees only and Arm partners with an NDA agreement.

Usage constraints

There are no usage constraints.

Configurations

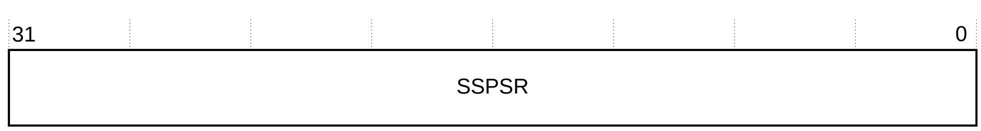
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_SSPSR bit assignments.

Figure B-2: TPIU_SSPSR bit assignments



The following table shows the TPIU_SSPSR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table B-3: TPIU_SSPSR bit assignments

Bits	Name	Function
[31:0]	SSPSR	Supported sizes of **TRACEDATA[3:0]**. The possible values are: `0b0001` Maximum 1-bit trace port. `0b0011` Maximum 2-bit trace port. `0b1011` Maximum 4-bit trace port.

B.2.2 TPIU_CSPSR, Current Port Size Register

TPIU_CSPSR shows the currently selected size of the trace data port, **TRACEDATA[3:0]**.

It has the same format as the TPIU_SSPSR register, but only one bit is set to show the currently selected port size. If a bit that is indicated as not supported in the TPIU_SSPSR is set in the TPIU_CSPSR, it can corrupt the output trace stream, in trace capture mode, and the trace patterns in pattern generation mode. If more than one bit is set, the port size is internally resolved to the highest order set bit. This register must not be modified while the trace port is still active, or without correctly stopping the formatter. If this happens, it can result in data not being aligned to the port width.

Usage constraints

There are no usage constraints.

Configurations

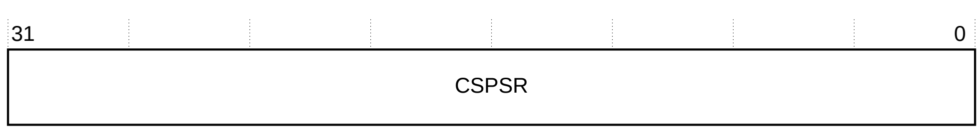
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_CSPSR bit assignments.

Figure B-3: TPIU_CSPSR bit assignments



The following table shows the TPIU_CSPSR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table B-4: TPIU_CSPSR bit assignments

Bits	Name	Function
[31:0]	CSPSR	Currently selected size of the trace data port **TRACEDATA[3:0]**. The possible values are: `0b0001` 1-bit trace port `0b0010` 2-bit trace port `0b1000` 4-bit trace port

B.2.3 TPIU_SPPR, Selected Pin Protocol Register

TPIU_SPPR selects which protocol is used by the TPIU for trace output.

Usage constraints

There are no usage constraints.

Configurations

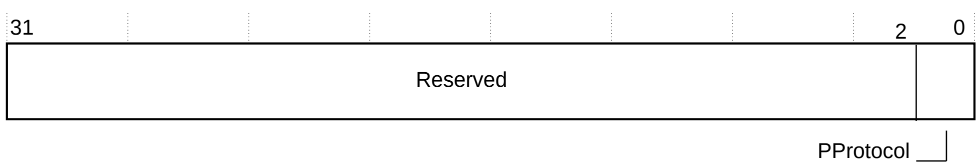
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_SPPR bit assignments.

Figure B-4: TPIU_SPPR bit assignments



The following table shows the TPIU_SPPR bit assignments.

Table B-5: TPIU_SPPR bit assignments

Bits	Name	Function
[31:2]	-	RES0
[1:0]	PProtocol	Pin protocol used for trace output. The options are: `0x0` Parallel port `0x1` SWO Manchester `0x2` SWO NRZ (UART)

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.4 TPIU_PSCR, Periodic Synchronization Counter Register

TPIU_PSCR determines the reload value of the Periodic Synchronization Counter. This counter enables the frequency of sync packets to be optimized to the trace capture buffer size.

Usage constraints

There are no usage constraints.

Configurations

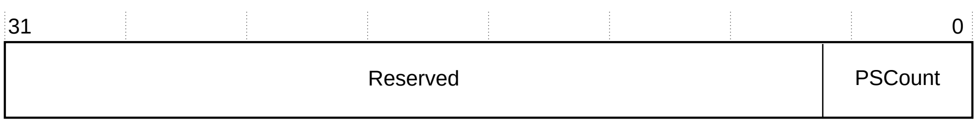
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_PSCR bit assignments.

Figure B-5: TPIU_PSCR bit assignments



The following table shows the TPIU_PSCR bit assignments.

Table B-6: TPIU_PSCR bit assignments

Bits	Name	Function
[31:5]	-	RAZ/WI
[4:0]	PSCount	Periodic Synchronization Count that determines the reload value of the Synchronization Counter. The Periodic Synchronization Counter counts up to a maximum of 2^{16} bytes, where the TPIU_PSCR.PSCount value determines the reload value of Synchronization Counter, as 2 to the power of the programmed value. The TPIU_PSCR.PSCount value has a range between `0b00111` and `0b10000`, any attempt to program register with a value smaller than the minimum value disables the Synchronization Counter. If the programmed reload value is greater than the maximum value, then the Periodic Synchronization Counter is reloaded with its maximum value and the TPIU will generate synchronization requests at this interval.

B.2.5 TPIU_ACPR, Asynchronous Clock Prescaler Register

TPIU_ACPR scales the Baud rate of the asynchronous output.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

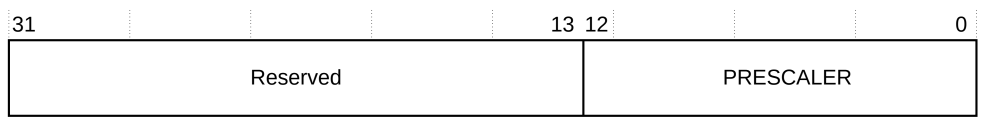
Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following figure shows the TPIU_ACPR bit assignments.

Figure B-6: TPIU_ACPR bit assignments



The following table shows the TPIU_ACPR bit assignments.

Table B-7: TPIU_ACPR bit assignments

Bits	Name	Function
[31:13]	-	Reserved. RAZ/SBZP.
[12:0]	PRESCALER	Divisor for TRACECLKIN is Prescaler + 1.

B.2.6 TPIU_FFSR, Formatter and Flush Status Register

TPIU_FFSR indicates the status of the TPIU formatter.

Usage constraints

There are no usage constraints.

Configurations

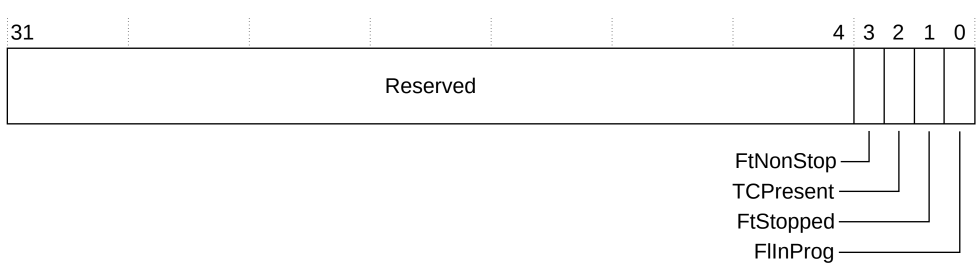
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_FFSR bit assignments.

Figure B-7: TPIU_FFSR bit assignments



The following table shows the TPIU_FFSR bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table B-8: TPIU_FFSR bit assignments

Bit	Name	Type	Description
[31:4]	Reserved	-	RES0
[3]	FtNonStop	RO	Formatter cannot be stopped
[2]	TCPresent	RO	This bit is always `0b0`.
[1]	FtStopped	RO	This bit is always `0b0`.
[0]	FlInProg	RO	Flush in progress. The values read can be: **0** When all the data received, before the flush is acknowledged, has been output on the trace port **1** When a flush is initiated

B.2.7 TPIU_FFCR, Formatter and Flush Control Register

TPIU_FFCR controls the TPIU formatter.

Usage constraints

There are no usage constraints.

Configurations

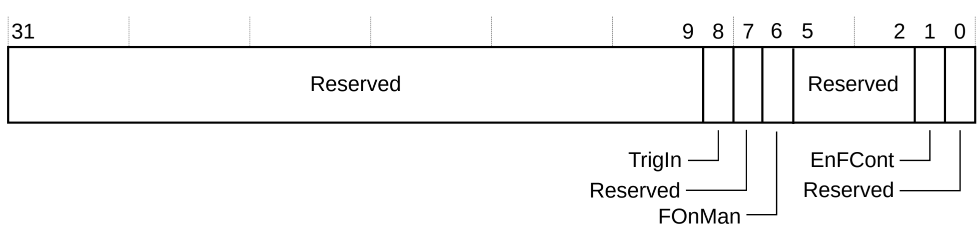
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_FFCR bit assignments.

Figure B-8: TPIU_FFCR bit assignments



The following table shows the TPIU_FFCR bit assignments.

Table B-9: TPIU_FFCR bit assignments

Bit	Name	Type	Description
[31:9]	Reserved	-	RES0
[8]	TrigIn	-	This bit Reads-As-One (RAO), specifying that triggers are inserted when a trigger pin is asserted.
[7]	Reserved	-	RES0

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Bit	Name	Type	Description
[6]	FOnMan	RW	Flush on manual. The options are: **0** When the flush completes. Set to 0 on a reset of the TPIU. **1** Generates a flush.
[5:2]	Reserved	-	**RES0**
[1]	EnFCont	RW	Enable continuous formatting. The options are: **0** Continuous formatting disabled. **1** Continuous formatting enabled.
[0]	Reserved	-	**RES0**

The TPIU can output trace data in a *Serial Wire Output* (SWO) format. See B.1.2 Serial Wire Output format on page 314.

Note

If TPIU_SPPR is set to select Trace Port Mode, the formatter is automatically enabled. If you then select one of the SWO modes, TPIU_FFCR reverts to its previously programmed value.

B.2.8 TPIU_TRIGGER, TPIU TRIGGER Register

The TPIU_TRIGGER register controls the integration test **TRIGGER** input.

Usage constraints

There are no usage constraints.

Configurations

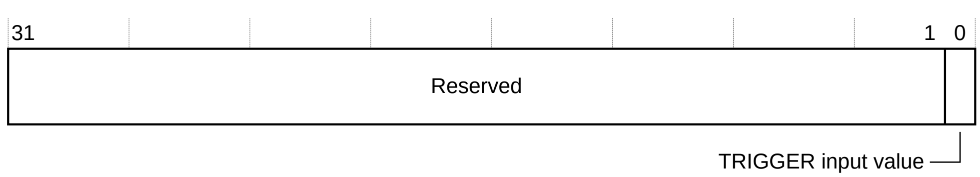
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_TRIGGER bit assignments.

Figure B-9: TPIU_TRIGGER bit assignments



The following table shows the TPIU_TRIGGER bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table B-10: TPIU_TRIGGER bit assignments

Bit	Name	Type	Description
[31:1]	Reserved	-	**RES0**
[0]	TRIGGER input value	RO	When read, this bit returns the **TRIGGER** input value.

B.2.9 ITFTTD0, Integration Test FIFO Test Data 0 Register

ITFTTD0 controls trace data integration testing.

Usage constraints

You must set bit[1] of TPIU_ITCTRL to use this register. See B.2.13 TPIU_ITCTRL, Integration Mode Control on page 327.

Configurations

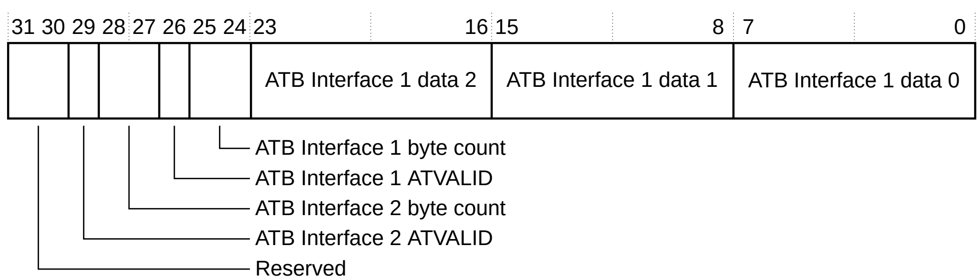
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the Integration Test FIFO Test Data 0 Register data bit assignments.

Figure B-10: ITFTTD0 bit assignments



The following table shows the ITFTTD0 bit assignments.

Table B-11: ITFTTD0 bit assignments

Bits	Name	Function
[31:30]	-	Reserved.
[29]	ATB Interface 2 ATVALID input	Returns the value of the ATB Interface 2 ATVALID signal.
[28:27]	ATB Interface 2 byte count	Number of bytes of ATB Interface 2 trace data since last read of this register.
[26]	ATB Interface 1 ATVALID input	Returns the value of the ATB Interface 1 ATVALID signal.
[25:24]	ATB Interface 1 byte count	Number of bytes of ATB Interface 1 trace data since last read of this register.
[23:16]	ATB Interface 1 data 2	ATB Interface 1 trace data. The TPIU discards this data when the register is read.
[15:8]	ATB Interface 1 data 1	
[7:0]	ATB Interface 1 data 0	

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.10 ITATBCTR2, Integration Test ATB Control Register 2

ITATBCTR2 controls integration test.

Usage constraints

You must set bit[0] of TPIU_ITCTRL to use this register. See B.2.13 TPIU_ITCTRL, Integration Mode Control on page 327.

Configurations

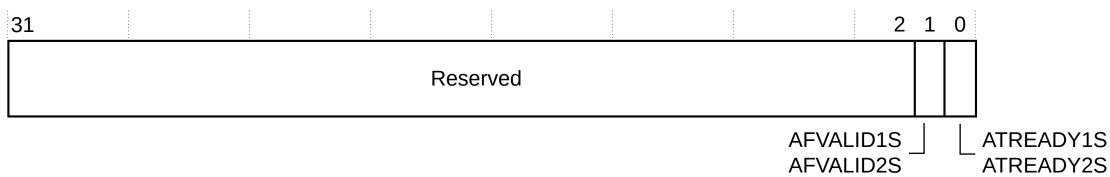
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the ITATBCTR2 bit assignments.

Figure B-11: ITATBCTR2 bit assignments



The following table shows the ITATBCTR2 bit assignments.

Table B-12: ITATBCTR2 bit assignments

Bits	Name	Function
[1]	AFVALID1S, AFVALID2S	This bit sets the value of both the ATB Interface 1 and 2 AFVALID outputs, if the TPIU is in integration test mode.
[0]	ATREADY1S, ATREADY2S	This bit sets the value of both the ATB Interface 1 and 2 ATREADY outputs, if the TPIU is in integration test mode.

B.2.11 ITFTTD1, Integration Test FIFO Test Data 1 Register

ITFTTD1 controls trace data integration testing.

Usage constraints

You must set bit[1] of TPIU_ITCTRL to use this register. See B.2.13 TPIU_ITCTRL, Integration Mode Control on page 327.

Configurations

Available in all configurations.

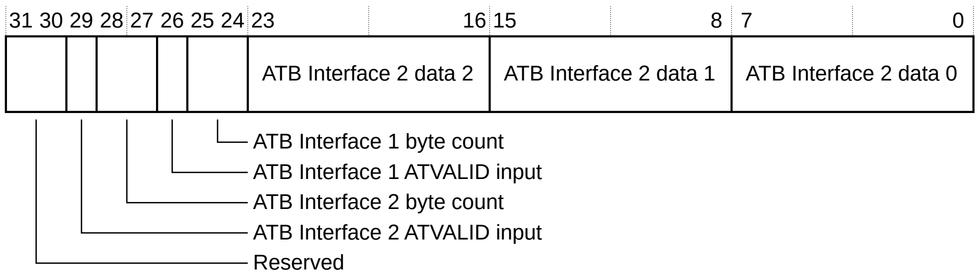
Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the ITFTTD1 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure B-12: ITFTTD1 bit assignments



The following table shows the ITFTTD1 bit assignments.

Table B-13: ITFTTD1 bit assignments

Bits	Name	Function
[31:30]	-	Reserved.
[29]	ATB Interface 2 ATVALID input	Returns the value of the ATB Interface 2 ATVALID signal.
[28:27]	ATB Interface 2 byte count	Number of bytes of ATB Interface 2 trace data since last read of this register.
[26]	ATB Interface 1 ATVALID input	Returns the value of the ATB Interface 1 ATVALID signal.
[25:24]	ATB Interface 1 byte count	Number of bytes of ATB Interface 1 trace data since last read of this register.
[23:16]	ATB Interface 2 data 2	ATB Interface 2 trace data. The TPIU discards this data when the register is read.
[15:8]	ATB Interface 2 data 1	
[7:0]	ATB Interface 2 data 0	

B.2.12 ITATBCTR0, Integration Test ATB Control 0 Register

ITATBCTR0 is used for integration test.

Usage constraints

There are no usage constraints.

Configurations

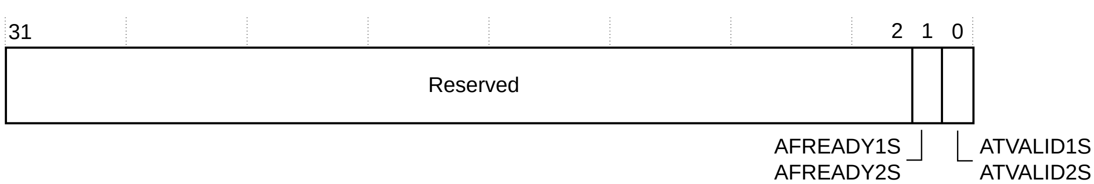
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the ITATBCTR0 bit assignments.

Figure B-13: ITATBCTR0 bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table shows the ITATBCTR0 bit assignments.

Table B-14: ITATBCTR0 bit assignments

Bits	Name	Function
[1]	AFREADY1S, AFREADY2S	A read of this bit returns the value of AFREADY1S OR-gated with AFREADY2S.
[0]	ATVALID1S, ATVALID2S	A read of this bit returns the value of ATVALID1S OR-gated with ATVALID2S.

B.2.13 TPIU_ITCTRL, Integration Mode Control

TPIU_ITCTRL specifies normal or integration mode for the TPIU.

Usage constraints

There are no usage constraints.

Configurations

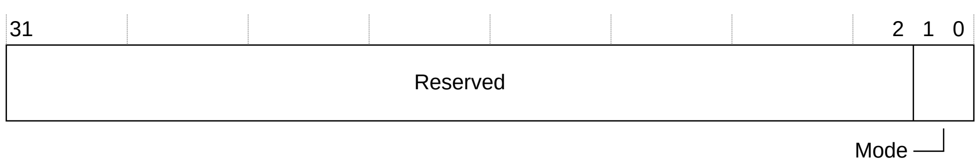
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_ITCTRL bit assignments.

Figure B-14: TPIU_ITCTRL bit assignments



The following table shows the TPIU_ITCTRL bit assignments.

Table B-15: TPIU_ITCTRL bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1:0]	Mode	Specifies the current mode for the TPIU: **0b00** Normal mode. **0b01** Integration test mode. **0b10** Integration data test mode. **0b11** Reserved. In integration data test mode, the trace output is disabled, and data can be read directly from each input port using the integration data registers.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.14 CLAIMSET, Claim Tag Set Register

The CLAIMSET register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

Usage constraints

See B.2 TPIU register summary on page 315 for more information.

Configurations

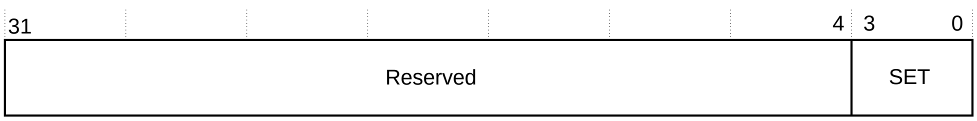
This register is always implemented.

Attributes

This is a 32-bit register.

The following figure shows the CLAIMSET bit assignments.

Figure B-15: CLAIMSET bit assignments



The following table describes the CLAIMSET bit assignments.

Table B-16: CLAIMSET bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	**RES0**
[3:0]	SET	RW	The options are: **Write 0** No effect. **Write 1** Set the claim tag for bit[n]. **Read 0** The claim tag that is represented by bit[n] is not implemented. **Read 1** The claim tag that is represented by bit[n] is implemented.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.15 CLAIMCLR, Claim Tag Clear Register

The CLAIMCLR register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

Usage constraints

See B.2 TPIU register summary on page 315 for more information.

Configurations

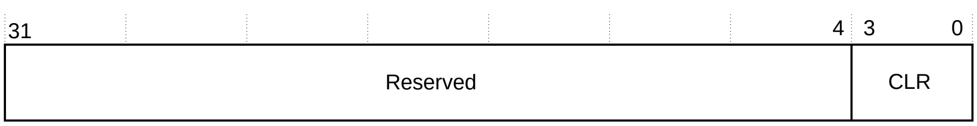
This register is always implemented.

Attributes

This is a 32-bit register.

The following figure shows the CLAIMCLR bit assignments.

Figure B-16: CLAIMCLR bit assignments



The following table describes the CLAIMCLR bit assignments.

Table B-17: CLAIMCLR bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CLR	RW	The options are: **Write 0** No effect. **Write 1** Clear the claim tag for bit[n]. **Read 0** The claim tag that is represented by bit[n] is not set. **Read 1** The claim tag that is represented by bit[n] is set.

B.2.16 TPIU_DEVID, Device Configuration Register

TPIU_DEVID indicates the functions that are provided by the TPIU for use in the topology detection.

Usage constraints

There are no usage constraints.

Configurations

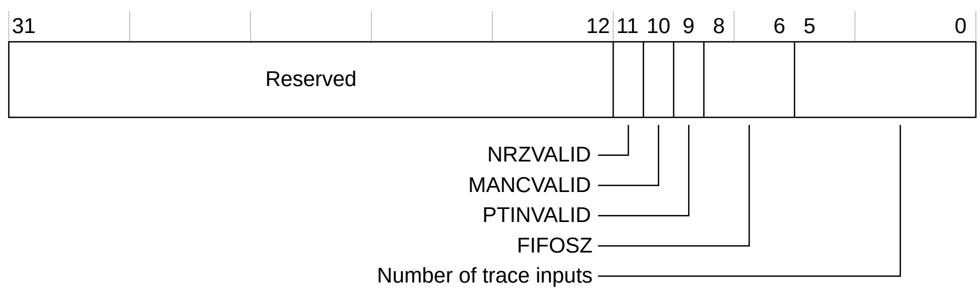
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_DEVID bit assignments.

Figure B-17: TPIU_DEVID bit assignments



The following table shows the TPIU_DEVID bit assignments.

Table B-18: TPIU_DEVID bit assignments

Bits	Name	Function
[31:12]	-	Reserved.
[11]	NRZVALID	Indicates support for SWO using UART/NRZ encoding. Always RAO. The output is supported.
[10]	MANCVALID	Indicates support for SWO using Manchester encoding. Always RAO. The output is supported.
[9]	PTINVALID	Indicates support for parallel trace port operation. Always RAZ. Trace data and clock modes are supported.
[8:6]	FIFOSZ	Indicates the implemented size of the TPIU output FIFO for trace data: `0b010` Four bytes.
[5:0]	Number of trace inputs	Specifies the number of trace inputs: `0b000000` One input. `0b000001` Two inputs.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

B.2.17 TPIU_DEVTYPE, Device Type Identifier Register

TPIU_DEVTYPE provides a debugger with information about the component when the Part Number field is not recognized. The debugger can then report this information.

Usage Constraints

There are no usage constraints.

Configurations

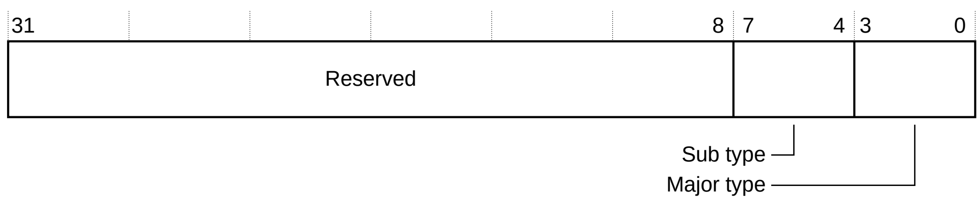
Available in all configurations.

Attributes

See Table B-2: TPIU IMPLEMENTATION DEFINED register summary on page 315.

The following figure shows the TPIU_DEVTYPE bit assignments.

Figure B-18: TPIU_DEVTYPE bit assignments



The following table shows the TPIU_DEVTYPE bit assignments.

Table B-19: TPIU_DEVTYPE bit assignments

Bits	Name	Function
[31:8]	-	Reserved.
[7:4]	Sub type	`0x1` Identifies the classification of the debug component.
[3:0]	Major type	`0x1` Indicates this device is a trace sink and specifically a TPIU.

B.2.18 TPIU_PIDR4, Peripheral Identification Register 4

The TPIU_PIDR4 register provides information about the memory size and JEP106 continuation code that the *Trace Port Interface Unit* (TPIU) component uses.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Configurations

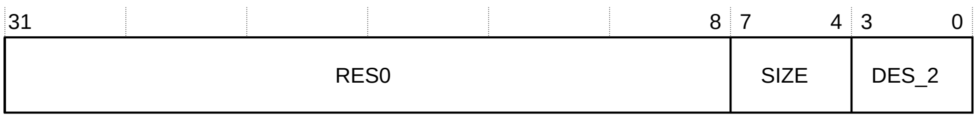
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See 19.2 CTI register summary on page 268 for more information.

The following figure shows the TPIU_PIDR4 bit assignments.

Figure B-19: TPIU_PIDR4 bit assignments



The following table describes the TPIU_PIDR4 bit assignments.

Table B-20: TPIU_PIDR4 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	SIZE	RO	This field indicates the memory size that the TPIU uses. This field returns `0x0` indicating that the component uses an **UNKNOWN** number of 4KB blocks. The reset value of this field is `0x0`.
[3:0]	DES_2	RO	JEP106 continuation code. Together with TPIU_PIDR2.DES_1 and TPIU_PIDR1.DES_0, they indicate the designer of the component, not the implementer, except where the two are the same. The reset value of this field is `0x4`.

B.2.19 TPIU_PIDR5, Peripheral Identification Register 5

The TPIU_PIDR5 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

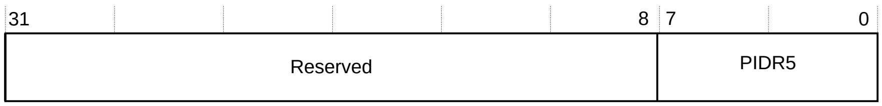
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR5 bit assignments.

Figure B-20: TPIU_PIDR5 bit assignments



The following table describes the TPIU_PIDR5 bit assignments.

Table B-21: TPIU_PIDR5 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PIDR5	RO	**RES0**.

B.2.20 TPIU_PIDR6, Peripheral Identification Register 6

The TPIU_PIDR6 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

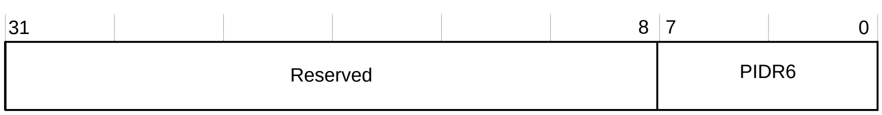
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR6 bit assignments.

Figure B-21: TPIU_PIDR6 bit assignments



The following table describes the TPIU_PIDR6 bit assignments.

Table B-22: TPIU_PIDR6 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[7:0]	PIDR6	RO	**RES0**.

B.2.21 TPIU_PIDR7, Peripheral Identification Register 7

The TPIU_PIDR7 register is reserved.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

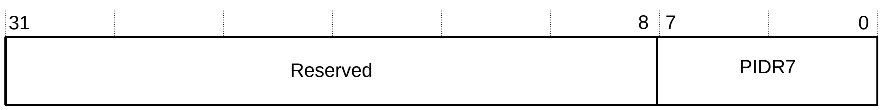
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR7 bit assignments.

Figure B-22: TPIU_PIDR7 bit assignments



The following table describes the TPIU_PIDR7 bit assignments.

Table B-23: TPIU_PIDR7 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PIDR7	RO	**RES0**.

B.2.22 TPIU_PIDR0, Peripheral Identification Register 0

The TPIU_PIDR0 register indicates the TPIU component part number.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the TPIU is included.

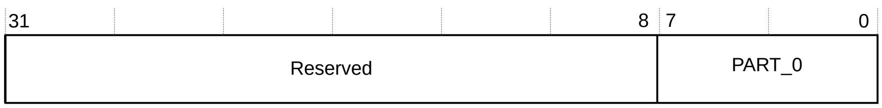
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR0 bit assignments.

Figure B-23: TPIU_PIDR0 bit assignments



The following table describes the TPIU_PIDR0 bit assignments.

Table B-24: TPIU_PIDR0 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PART_0	RO	This field indicates the part number. When taken together with TPIU_PIDR1.PART_1, it indicates the component. The part number is selected by the designer of the component. The reset value of this field is `0x00100011`.

B.2.23 TPIU_PIDR1, Peripheral Identification Register 1

The TPIU_PIDR1 register indicates the TPIU component JEP106 continuation code and part number.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the TPIU is included.

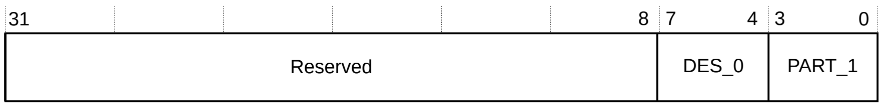
Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR1 bit assignments.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure B-24: TPIU_PIDR1 bit assignments



The following table describes the TPIU_PIDR1 bit assignments.

Table B-25: TPIU_PIDR1 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:4]	DES_0	RO	This field indicates the JEP106 identification code, bits[3:0]. Together, with TPIU_PIDR4.DES_2 and TPIU_PIDR2.DES_1, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is `0xB`.
[3:0]	PART_1	RO	This field indicates the part number, bits[11:8]. Taken together with TPIU_PIDR0.PART_0 it indicates the component. The part number is selected by the designer of the component. The reset value is `0xD`.

B.2.24 TPIU_PIDR2, Peripheral Identification Register 2

The TPIU_PIDR2 register indicates the TPIU component revision number, JEDEC value, and part of the JEP106 continuation code.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

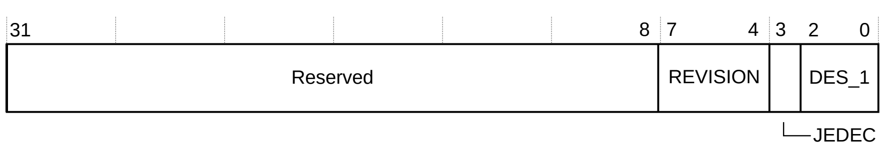
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR2 bit assignments.

Figure B-25: TPIU_PIDR2 bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the TPIU_PIDR2 bit assignments.

Table B-26: TPIU_PIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:4]	REVISION	RO	This field indicates the revision number of the TPIU component. It is an incremental value starting at `0x0` for the first design. The reset value is `0x0`.
[3]	JEDEC	RO	This field is always 1, indicating that a JEDEC assigned value is used.
[2:0]	DES_1	RO	This field is the JEP106 identification code, bits[6:4]. Together, with TPIU_PIDR4.DES_2 and TPIU_PIDR1.DES_0, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is `0b011`.

B.2.25 TPIU_PIDR3, Peripheral Identification Register 3

The TPIU_PIDR3 register indicates minor errata fixes of the TPIU component and if you have modified the behavior of the component.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

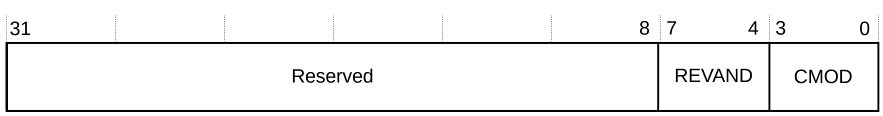
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_PIDR3 bit assignments.

Figure B-26: TPIU_PIDR3 bit assignments



The following table describes the TPIU_PIDR3 bit assignments.

Table B-27: TPIU_PIDR3 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:4]	REVAND	RO	This field indicates minor errata fixes specific to this design, for example metal fixes after implementation. In most cases this field is `0x0`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Field	Name	Type	Description
[3:0]	CMOD	RO	Customer modified. Where the component is reusable IP, this value indicates whether you have modified the behavior of the component. In most cases, this field is `0x0`.

B.2.26 TPIU_ CIDR0, Component Identification Register 0

The TPIU_CIDR0 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

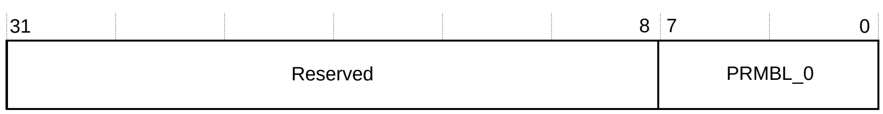
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_CIDR0 bit assignments.

Figure B-27: TPIU_CIDR0 bit assignments



The following table describes the TPIU_CIDR0 bit assignments.

Table B-28: TPIU_CIDR0 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PRMBL_0	RO	Preamble. This field returns `0x0D`.

B.2.27 TPIU_ CIDR1, Component Identification Register 1

The TPIU_CIDR1 register indicates the component class and preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the TPIU is included.

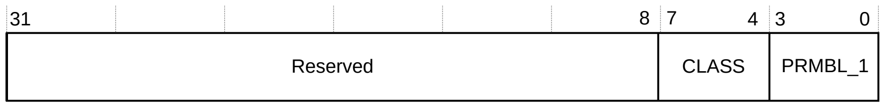
Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_CIDR1 bit assignments.

Figure B-28: TPIU_CIDR1 bit assignments



The following table describes the TPIU_CIDR1 bit assignments.

Table B-29: TPIU_CIDR1 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:4]	CLASS	RO	Component class. Returns `0x9`, indicating this is a CoreSight™ component.
[3:0]	PRMBL_1	RO	Preamble. This field returns `0x0`.

B.2.28 TPIU_ CIDR2, Component Identification Register 2

The TPIU_CIDR2 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

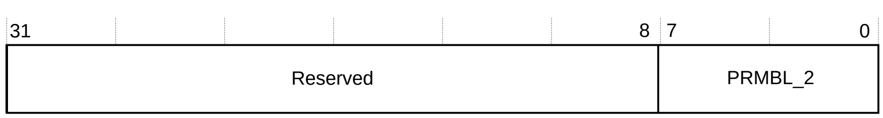
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_CIDR2 bit assignments.

Figure B-29: TPIU_CIDR2 bit assignments



Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table describes the TPIU_CIDR2 bit assignments.

Table B-30: TPIU_CIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PRMBL_2	RO	Preamble. This field returns `0x05`.

B.2.29 TPIU_ CIDR3, Component Identification Register 3

The TPIU_CIDR3 register indicates the preamble.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

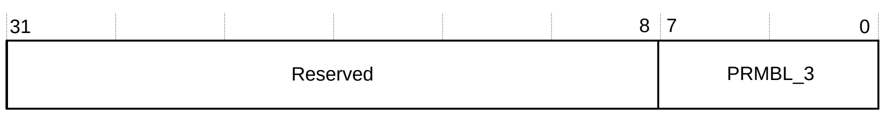
This register is always implemented when the TPIU is included.

Attributes

This is a 32-bit register. See B.2 TPIU register summary on page 315 for more information.

The following figure shows the TPIU_CIDR3 bit assignments.

Figure B-30: TPIU_CIDR3 bit assignments



The following table describes the TPIU_CIDR3 bit assignments.

Table B-31: TPIU_CIDR3 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	**RES0**.
[7:0]	PRMBL_3	RO	Preamble. This field returns `0xB1`.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Appendix C Signal descriptions

This appendix describes the Cortex®-M85 processor signals.

C.1 Clock and clock enable signals

The following table shows the Cortex®-M85 processor clock and clock enable signals.

Table C-1: Clock and clock enable signals

Signal name	Direction	Description
CLKIN	Input	Primary processor clock. This is gated internally for functional units when required depending on the operating mode of the processor.
DBGCLK	Input	Clock driving the majority of the debug and trace logic in the processor.
SSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the secure system SysTick clock.
NSSTCLKEN	Input	Synchronous enable that is used with **CLKIN** to derive the Non-secure system SysTick clock.
CLKINDCLS	Input	Primary clock for the redundant processor logic. This signal must be synchronous to **CLKIN**. **Note:** This signal is reserved for future use and must be tied off.
IWICCLKDCLS	Input	Clock for the redundant IWIC logic. This clock must be synchronous to **IWICCLK**. **Note:** This signal is reserved for future use and must be tied off.
IWICCLK	Input	This signal is the IWIC clock.

C.2 Reset signals

The following table shows the Cortex®-M85 processor reset signals.

Table C-2: Reset signals

Signal name	Direction	Description
nPORESET	Input	Cold reset. The **nPORESET** signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.
nSYSRESET	Input	System reset. This signal resets non-debug logic and all memory interfaces from the core power domain (PDCORE). The **nSYSRESET** signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.
nDBGRESET	Input	Debug reset that resets all logic in the debug power domain (PDDEBUG). This reset must be asserted when PDDEBUG is powered down.
nIWICRESET	Input	This is an active-LOW IWIC reset signal. This signal is internally synchronized to **IWICCLK**. The **nIWICRESET** signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
nMBISTRESET	Input	Production MBIST reset.

C.3 Static configuration signals

The following table shows the Cortex®-M85 processor static configuration signals.

The configuration signals in the following table can only be changed at Cold reset with **nPORESET** asserted. They are intended to be static configuration signals that are fixed for a given integration of the processor.

Table C-3: Static configuration signals

Signal name	Direction	Description
CFGITCMSZ[3:0]	Input	Size of the *Instruction Tightly Coupled Memory* (ITCM) region encoded as: **CFGITCMSZ = `0b0000`** ITCM is not implemented. **CFGITCMSZ > `0b0001`** $2^{CFGDTCMSZ-1}$KB • The minimum size of *Tightly Coupled Memory* (TCM) is 4KB and the maximum size is 16MB. Setting **CFGITCMSZ** to `0b0001` results in **UNPREDICTABLE** behavior. • The **CFGITCMSZ** input signal sets the ITCM size. The `ITGUMAXBLKS` parameter constraints the maximum ITCM size that can be used. Therefore, the `ITGUMAXBLKS` must be set to be large enough to accommodate the anticipated ITCM size that might be used in the system.
CFGDTCMSZ[3:0]	Input	Size of the *Data Tightly Coupled Memory* (DTCM) region encoded as: **CFGDTCMSZ = `0b0000`** DTCM is not implemented. **CFGDTCMSZ > `0b0010`** $2^{CFGDTCMSZ-1}$KB • The **CFGDTCMSZ** input signal sets the DTCM size. The `DTGUMAXBLKS` parameter constraints the maximum DTCM size that can be used. Therefore, the `DTGUMAXBLKS` must be set to be large enough to accommodate the anticipated DTCM size that might be used in the system. • The minimum size of the TCM is 4KB and the maximum size is 16MB. Setting **CFGDTCMSZ** to `0b0001` or `0b0010` results in **UNPREDICTABLE** behavior.
CFGPAHBSZ[2:0]	Input	Size of the *Peripheral AHB* (P-AHB) peripheral port memory region: **`0bb000`** P-AHB disabled. **`0bb001`** 64MB **`0bb010`** 128MB **`0bb011`** 256MB **`0bb100`** 512MB Setting **CFGPAHBSZ** to any other value results in **UNPREDICTABLE** behavior.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
CFGMEMALIAS[4:0]	Input	Memory address alias bit for the ITCM, DTCM, and P-AHB regions. The address bit used for the memory alias is determined by: `0b00001` Alias bit = 24 `0b00010` Alias bit = 25 `0b00100` Alias bit = 26 `0b01000` Alias bit = 27 `0b10000` Alias bit = 28 `0b00000` No alias. TCM security gating is disabled. Setting **CFGMEMALIAS** to any other value is invalid, and results in **UNPREDICTABLE** behavior. For more information on memory aliasing and IDAU/SAU configuration, see 9.8.1 Memory aliasing and IDAU/SAU configuration on page 147.
CFGFPU	Input	If the *Floating-point Unit* (FPU) is configured, enables support for floating-point operation.
CFGMVE[1:0]	Input	If *M-profile Vector Extension* (MVE) in configured, enables support for MVE: `0b00` No MVE. `0b01` If **CFGFPU** is set to 0 or FPU is set to FALSE, Integer Vector MVE *Instruction Set Architecture* (ISA) is supported. `0b10` If **CFGFPU** is set to 1 and FPU is set to TRUE, integer and floating-point vector MVE ISA is supported.
CFGBIGEND	Input	Selects the data endian format: **0** Little-endian (LE). **1** Byte-invariant big-endian (BE8).
MPUNSDISABLE	Input	Disables support for the Non-secure MPU region.
MPUSDISABLE	Input	Disables support for the Secure MPU region.
SAUDISABLE	Input	Disables support of *Security Attribution Unit* (SAU).
CFGSSTCALIB[25:0]	Input	Secure SysTick calibration configuration: **CFGSTCALIB[23:0]** TENMS **CFGSTACLIB[24]** SKEW **CFGSTCALIB[25]** NOREF
CFGNSSTCALIB[25:0]	Input	Non-secure SysTick calibration configuration: **CFGNSTCALIB[23:0]** TENMS **CFGNSTCALIB[24]** SKEW **CFGNSTCALIB[25]** NOREF
CFGPACBTI	Input	Enables support for the *Pointer Authentication and Branch Target Identification* (PACBTI) Extension: `0bb0` PACBTI disabled. `0bb1` PACBTI enabled.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

C.4 Reset configuration signals

The following table shows the Cortex®-M85 processor reset configuration signals. These signals are sampled at deassertion of Warm reset or Cold reset, and their values can change out of reset. The reset configuration signals can be used more dynamically than the static configuration signals.

Table C-4: Reset configuration signals

Signal name	Direction	Description
INITSVTOR[31:7]	Input	This signal indicates the Secure vector table offset address out of reset, VTOR_S.TBLOFF[31:7]. For more information on VTOR_S, see the *Arm®v8-M Architecture Reference Manual*.
INITNSVTOR[31:7]	Input	This signal indicates the Non-secure vector table offset address out of reset, VTOR_NS.TBLOFF[31:7]. For more information on VTOR_NS, see the *Arm®v8-M Architecture Reference Manual*.
INITTCMEN[1:0]	Input	*Tightly Coupled Memory* (TCM) enable initialization out of reset: **Bit[0] is HIGH:** *Instruction Tightly Coupled Memory* (ITCM) is enabled. **Bit[1] is HIGH:** *Data Tightly Coupled Memory* (DTCM) is enabled. This signal controls the reset value of ITCMCR.EN and DTCMCR.EN bits. For more information on ITCMCR and DTCMCR, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*.
INITPAHBEN	Input	P-AHB enable initialization out of reset: **HIGH** P-AHB is enabled. **LOW** P-AHB disabled. For more information on PAHBCR, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*.
INITECCEN	Input	TCM and L1 cache *Error Correcting Code* (ECC) enable out of reset. **HIGH** ECC is enabled. **LOW** ECC is disabled. If ECC is not configured in the processor, this signal has no effect on the processor. ECC must not be enabled dynamically when the processor is in the Memory retention mode (MEM_RET) power mode. This is because the L1 cache is not automatically invalidated with the Memory retention mode power mode is switched on. This results in inconsistent ECC information that is relative to the data that is retained in the cause. This results in an ECC error.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

C.5 Cache initialization signal

The data and instruction caches can be automatically initialized when enabled at reset or if the PDRAMS power domain is enabled during runtime. This functionality can be disabled if required using the **INITL1RSTDIS** signal. The following table describes the **INITL1RSTDIS** signal.

Table C-5: Cache initialization signal

Signal name	Direction	Description
INITL1RSTDIS	Input	Disable L1 cache invalidation out of reset. **HIGH** Disable automatic invalidation of the L1 cache. **LOW** Enable automatic invalidation of the L1 cache that occurs in the following cases: • The P-Channel is used to turn on the PDCORE domain. Power mode transitions from OFF to ON or OFF to EPU_OFF. Invalidation does not occur on transitions from OFF to MEM_RET or MEM_RET to ON. • **nSYSRESET** is asserted when the PDRAMS are powered on, that is, when the processor is in either of the following: ◦ The power modes ON (cache) or EPU_OFF (cache). Arm does not recommend that you assert **nSYSRESET** when the processor is ON (Cache) or EPU_OFF (Cache) because this can cause a system error. ◦ The WARM_RST power mode with PDRAMS on. • The P-Channel is used to move the power mode from ON (no cache) to ON (cache). **Note:** • If the P-Channel is used to control the processor power mode selection, then this signal must be tied LOW unless valid cache RAM content is required to be preserved after WARM_RST. • If **INITECCEN** is HIGH, this signal must be LOW on reset unless the content of the instruction and data cache tag RAMs is guaranteed to be valid. For more information on the P-Channel and power modes, see 7 Power management on page 115.

C.6 Instruction execution control signals

The following table shows the instruction execution control signals that must be connected in your *System on Chip* (SoC) design.

Table C-6: Instruction execution control signals

Signal name	Direction	Description
CPUWAIT	Input	Stall the core out of reset.
CURRNS	Output	Current Security state of the Cortex®-M85 processor: **HIGH** Processor is in Non-secure state. **LOW** Processor is in Secure state.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
CURRPC[31:1]	Output	This signal indicates the address of the instruction that is currently executing on the Cortex®-M85 processor or address of the last instruction to execute during an exception entry or return. For dual issue instructions, this signal indicates the address of the older instruction with the lowest address. **Note:** **CURRNS** indicates the Security state of the executing instruction.
FAULTSTAT[42:0]	Output	This signal is asserted when the processor detects a fault while an exception is in progress. The signal encodes all the following Fault Status Registers: **FAULTSTAT[42:35]** SFSR[7:0] **FAULTSTAT[34]** HFSR.DEBUGEVT **FAULTSTAT[33]** HFSR.FORCED **FAULTSTAT[32]** HFSR.VECTTBL **FAULTSTAT[31:16]** UFSR **FAULTSTAT[15:8]** BFSR **FAULTSTAT[7:0]** MMFSR **Note:** This signal is not fully synchronous with the detection of the fault inside the processor.

C.7 Instruction Tightly Coupled Memory interface signals

The following table shows the Cortex®-M85 processor *Instruction Tightly Coupled Memory* (ITCM) interface signals. If you do not use the ITCM in your SoC, you must tie all the ITCM interface input signals to LOW.

Table C-7: ITCM interface signals

Signal name	Direction	Phase	Description
ITCMADDR[23:3]	Output	Address	Transfer address for reads and writes All ITCM accesses are 32-bit aligned. If necessary, the processor selects read data based on the full address.
ITCMCS	Output	Address	RAM chip select
ITCMPRIV	Output	Address	Privilege level of access: **0** User access **1** Privileged access
ITCMWR	Output	Address	RAM write enable: **0** Read access request **1** Write-Access request Valid when **ITCMCS** is HIGH.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Phase	Description
ITCMBYTEWR[8:0]	Output	Address	Byte write strobes **n<8** Bit[n] is to indicate data bits [8n+7:8n] **n=8** Bit[n] is to indicate that *Error Correcting Code* (ECC) information is written in ITCMWDATA[71:64] This signal is valid when **ITCMCS** is HIGH. **Note:** If **ITCMWR** is 0, **ITCMBYTEWR** = 0.
ITCMWDATA[71:0]	Output	Address	**ITCMWDATA[63:0]** Write data (64-bits) **ITCMWDATA[71:64]** ECC information (8-bits) **ITCMBYTEWR** defines validity of this signal on a byte-wise basis, otherwise, memory ignores this signal. If ECC is not configured, **ITCMWDATA[71:64]** can be left unconnected.
ITCMMASTER[3:0]	Output	Address	Encodes the requestor of the current access: `0b0000` Instruction fetch `0b0001` Data that is read from software on the processor `0b0010` Vector fetch on exception entry `0b0011` Read from *System AHB* (S-AHB) `0b0100` Debugger read `0b0101` *Memory Built-In Self Test* (MBIST) access `0b1001` Data write from software on the processor, including *Read Modify Write* (RMW) read access `0b1011` Debugger write `0b1100` ECC correction `0b1101` Stack pointer vector fetch, indicating that the TCM access is associated with reading the initial stack pointer from the reset vector `0b1110` Write from S-AHB, including RMW read access Can be used to monitor debug requests or used to change the behavior of TCM accesses for debug.
ITCMRDATA[71:0]	Input	Response	**ITCMRDATA[63:0]** Read data (64-bits) **ITCMRDATA[71:64]** ECC information (8-bits) All data bytes are valid on the last cycle of a read response phase. The processor ignores this signal on all other cycles.
ITCMWAIT	Input	Response	Wait signal to extend the current response phase: **0** Complete phase **1** Extend phase
ITCMERR	Input	Response	Error indication for the current transaction, valid on the last cycle of the response phase. **0** No error **1** Error

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

C.8 Data Tightly Coupled Memory interface signals

The following table shows the Cortex®-M85 processor *Data Tightly Coupled Memory* (DTCM) interface signals. If you are not using DTCM in your SoC, you must tie all the DTCM interface input signals to LOW.

Table C-8: DTCM interface signals

Signal name	Direction	Phase	Description
D*TCMADDR[23:4]	Output	Address	Transfer address for both reads and writes. All DTCM accesses are 32-bit aligned. The processor selects read data as required based on the full address.
D*TCMCS	Output	Address	RAM chip select.
D*TCMPRIV	Output	Address	Privilege level of access: **0** User access. **1** Privileged access.
D*TCMWR	Output	Address	RAM write enable: **0** Read access request. **1** Write access request. Valid when **D*TCMCS** is HIGH.
D*TCMBYTEWR[4:0]	Output	Address	Byte write strobes. **n<4** Bit[n] is to indicate data bits [8n+7:8n]. **n=4** Bit[n] is to indicate that *Error Correcting Code* (ECC) information is written in D*TCMWDATA[38:32]. This signal is valid when **D*TCMCS** is HIGH. **Note:** If **D*TCMWR** is 0, **D*TCMBYTEWR** is `0x0`.
D*TCMWDATA[38:0]	Output	Address	**D*TCMWDATA[31:0]** Write data (32-bits). **D*TCMWDATA[38:32]** *Error Correcting Code* (ECC) information (7-bits). **D*TCMBYTEWR** defines validity of this signal on a byte-wise basis, otherwise memory ignores this signal. If ECC is not configured, **D*TCMWDATA[38:32]** can be left unconnected.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Phase	Description
D*TCMMASTER[3:0]	Output	Address	Encodes the requestor of the current access: `0b0000` Instruction fetch. `0b0001` Data that is read from software on the processor. `0b0010` Vector fetch on exception entry. `0b0011` Read from *Slave AHB* (S-AHB). `0b0100` Debugger read. `0b0101` *Memory Built-In Self Test* (MBIST) access. `0b1001` Data write from software on the processor, including *Read Modify Write* (RMW) read access. `0b1011` Debugger write. `0b1100` ECC correction. `0b1101` Stack pointer vector fetch, indicating that the TCM access is associated with reading the initial stack pointer from the reset vector. `0b1110` Write from S-AHB including RMW read access. Can be used to monitor debug requests or used to change the behavior of TCM accesses for debug.
D*TCMRDATA[38:0]	Input	Response	**D*TCMRDATA[31:0]** Read data (32-bits). **D*TCMRDATA[38:32]** ECC information (7-bits). All data bytes are valid on the last cycle of a read response phase. The processor ignores this signal on all other cycles.
D*TCMWAIT	Input	Response	Wait signal to extend the current data phase: **0** Complete phase. **1** Extend phase.
D*TCMERR	Input	Response	Error indication for the current transaction, valid on the last cycle of the response phase. **LOW** No error. **HIGH** Error.

C.9 M-AXI interface signals

The *Master AXI* (M-AXI) interface implements the standard set of AMBA® 5 AXI read and write channel signals.

The following table shows the M-AXI master interface signals. For more information on the AMBA AXI signals, see the *AMBA® AXI and ACE Protocol Specification*.

Table C-9: M-AXI interface signals

Signal name	Direction	Description
ACLKEN	Input	Clock enable for the AXI port. Supports semi-synchronous operation of the interface relative to the processor clock. **Note:** **ACLKEN** can be used to clock all other M-AXI signals at an integer division of the processor clock. This includes support for timing the interface at n:1 for all other signals.
AWAKEUP	Output	Indicates that the master starts a transaction and sends it to the interconnect.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
AWVALID	Output	Write address valid signal.
AWADDR[31:0]	Output	Write address signal.
AWBURST[1:0]	Output	Write burst type signal.
AWLEN[2:0]	Output	Write burst length signal.
AWSIZE[1:0]	Output	Write burst size signal.
AWLOCK	Output	Write lock type signal.
AWPROT[2:0]	Output	Write protection type signal.
AWREADY	Input	Write address ready signal.
AWID[1:0]	Output	Write request ID signal. `0b00` Writes to Normal Non-cacheable memory and all store-exclusive transactions. `0b01` Writes to cacheable memory. `0b10` Writes to Device memory. `0b11` Cache line evictions.
AWCACHE[3:0]	Output	Outer Cacheability attributes. For more information on the encoding of this signal, see the *AMBA® AXI and ACE Protocol Specification*.
AWINNER[3:0]	Output	Inner Cacheability attributes. The encoding is identical to **AWCACHE[3:0]**. For more information on the encoding of **AWCACHE[3:0]** signal, see the *AMBA® AXI and ACE Protocol Specification*.
AWDOMAIN[1:0]	Output	Inner and outer Shareability attributes as defined in the active memory map. `0b00` Non-shareable `0b01` Reserved `0b10` Inner Shareable or Outer Shareable `0b11` System For more information on the encoding of this signal, see the *AMBA® AXI and ACE Protocol Specification*.
AWSPARSE	Output	Transaction might use sparse writes strobes. This signal indicates a write burst which might contain a beat which includes sparse data. That is, a beat which cannot be directly translated into an AHB transaction. If the signal is LOW, then the burst is guaranteed to be made up of contiguous and appropriately aligned data relative to data size.
AWMASTER	Output	Initiator of access. **0** Processor access. **1** Debugger access.
ARVALID	Output	Read address valid signal.
ARADDR[31:0]	Output	Read address signal.
ARBURST[1:0]	Output	Read burst type signal.
ARLEN[7:0]	Output	Read address burst length signal.
ARSIZE[1:0]	Output	Read burst size signal.
ARLOCK	Output	Read lock type signal.
ARPROT[2:0]	Output	Read protection type signal.
ARREADY	Input	Read address ready signal.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
ARID[2:0]	Output	Read request ID signal. `0b000` All accesses to Non-cacheable and Device memory regions (including bursts). `0b001` Data cache linefills from linefill buffer 0. `0b010` Data cache linefills from linefill buffer 1. `0b011` Data cache linefills from linefill buffer 2. `0b100` Data cache linefills from linefill buffer 3. `0b101`, `0b110` Reserved `0b111` Instruction fetch or instruction linefill.
ARCACHE[3:0]	Output	Outer Cacheability attributes. For more information on the encoding of this signal, see the *AMBA® AXI and ACE Protocol Specification*.
ARINNER[3:0]	Output	Inner Cacheability attributes. The encoding is identical to **ARCACHE[3:0]**. For more information on the encoding of **ARCACHE[3:0]** signal, see the *AMBA® AXI and ACE Protocol Specification*.
ARDOMAIN[1:0]	Output	Inner and Outer Shareability attributes as defined in the active memory map. `0b00` Non-shareable `0b01` Reserved `0b10` Inner Shareable and Outer Shareable `0b11` System For more information on the encoding of this signal, see the *AMBA® AXI and ACE Protocol Specification*.
ARMASTER	Output	Initiator of access. **0** Processor access. **1** Debugger access.
WID[1:0]	Output	Write data ID signal. Used to connect to AXI3 interconnect or slaves. Can be ignored for AXI4 or AXI5 interconnect or slaves. `0b00` Writes to Normal Non-cacheable memory and all store-exclusive transactions. `0b01` Writes to cacheable memory. `0b10` Writes to Device memory. `0b11` Cache line evictions.
WVALID	Output	Write data valid signal.
WLAST	Output	Indicates last transfer in a write burst.
WSTRB[7:0]	Output	Write byte lane strobes.
WDATA[63:0]	Output	Write data signal.
WPOISON	Output	Indicates that a set of data bytes has been corrupted.
WDATACHK[7:0]	Output	This signal can be used to detect, and potentially correct data bytes that might be corrupted.
WREADY	Input	Write data ready signal.
RVALID	Input	Read data valid signal.
RID[2:0]	Input	Read data ID. `0b000` All accesses to Non-cacheable and Device memory regions (including bursts). `0b001` Data cache linefills from linefill buffer 0. `0b010` Data cache linefills from linefill buffer 1. `0b011` Data cache linefills from linefill buffer 2. `0b100` Data cache linefills from linefill buffer 3. `0b101`, `0b110` Reserved `0b111` Instruction fetch or instruction linefill.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
RLAST	Input	Indicates last transfer in read data.
RDATA[63:0]	Input	Read data.
RRESP[1:0]	Input	Read data response.
RPOISON	Input	Indicates that a set of data bytes has been corrupted.
RDATACHK[7:0]	Input	This signal can be used to detect, and potentially correct data bytes that might be corrupted.
RREADY	Output	Read data ready signal.
BVALID	Input	Write response valid signal.
BID[1:0]	Input	Write response ID signal. `0b00` Writes to Normal Non-cacheable memory and all store-exclusive transactions. `0b01` Writes to cacheable memory. `0b10` Writes to Device memory. `0b11` Cache line evictions.
BRESP[1:0]	Input	Write response signal.
BREADY	Output	Write response ready signal.

C.9.1 M-AXI interface protection signals

The following table shows the M-AXI interface protection signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-10: M-AXI interface protection signals

Signal name	Direction	Description
ACLKENCHK	Input	Odd parity of **ACLKEN**.
AWAKEUPCHK	Output	Odd parity of **AWAKEUP**.
ARVALIDCHK	Output	Odd parity of **ARVALID**.
ARREADYCHK	Input	Odd parity of **ARREADY**.
ARADDRCHK[3:0]	Output	Odd parity of **ARADDR[31:0]** at 8-bit granularity.
ARIDCHK	Output	Odd parity of **ARID[2:0]**.
ARLENCHK	Output	Odd parity of **ARLEN[3:0].**
ARUSERCHK	Output	Odd parity of (**ARINNER[3:0]**, **ARMASTER**).
ARCTLCHK0	Output	Odd parity of (**ARSIZE[2:0]**, **ARBURST[1:0]**, **ARLOCK**, **ARPROT[2:0]**).
ARCTLCHK1	Output	Odd parity of **ARCACHE[3:0]**.
ARCTLCHK2	Output	Odd parity of **ARDOMAIN[1:0]**.
AWVALIDCHK	Output	Odd parity of **AWVALID**.
AWREADYCHK	Input	Odd parity of **AWREADY**.
AWADDRCHK[3:0]	Output	Odd parity of **AWADDR[31:0]** at 8-bit granularity.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
AWIDCHK	Output	Odd parity of **AWID[1:0]**.
AWLENCHK	Output	Odd parity of **AWLEN[3:0]**.
AWUSERCHK	Output	Odd parity of (**AWSPARSE**, **AWINNER[3:0]**, **AWMASTER**).
AWCTLCHK0	Output	Odd parity of (**AWSIZE[2:0]**, **AWBURST[1:0]**, **ARLOCK**, **ARPROT[2:0]**).
AWCTLCHK1	Output	Odd parity of (**AWCACHE[3:0]**, **AWPROT[2:0]**, **AWLOCK**).
AWCTLCHK2	Output	Odd parity of **AWDOMAIN[1:0]**.
RVALIDCHK	Input	Odd parity of **RVALID**.
RREADYCHK	Output	Odd parity of **RREADY**.
RIDCHK	Input	Odd parity of **RID[2:0]**.
RLASTCHK	Input	Odd parity of **RLAST**.
RRESPCHK	Input	Odd parity of **RRESP[1:0]**.
RPOISONCHK	Input	Odd parity of **RPOISON**.
WVALIDCHK	Output	Odd parity of **WVALID**.
WREADYCHK	Input	Odd parity of **WREADY**.
WSTRBCHK	Output	Odd parity of **WSTRB[7:0]**.
WIDCHK	Output	Odd parity of **WID[1:0]**.
WLASTCHK	Output	Odd parity of **WLAST**.
WPOISONCHK	Output	Odd parity of **WPOISON**.
BVALIDCHK	Input	Odd parity of **BVALID**.
BREADYCHK	Output	Odd parity of **BREADY**.
BIDCHK	Input	Odd parity of **BID[2:0]**.
BRESPCHK	Input	Odd parity of **BRESP[1:0]**.

C.10 S-AHB interface signals

The S-AHB interface provides direct access to the processor *Tightly Coupled Memory* (TCM) interfaces.

The following table shows the signals for the S-AHB interface.

Table C-11: S-AHB interface signals

Signal name	Direction	Description
HSELS	Input	This signal selects access to *Tightly Coupled Memory* (TCM) interfaces.
HTRANSS[1:0]	Input	Transfer type.
HBURSTS[2:0]	Input	Transfer burst length.
HADDRS[31:0]	Input	Transfer address and selected TCM interface.
HWRITES	Input	Write transfer.
HSIZES[2:0]	Input	Transfer size.
HWDATAS[63:0]	Input	Write data.
HWSTRBS[7:0]	Input	Write data byte lane strobes.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
HPROTS[6:0]	Input	Protection and outer memory attributes.
HNONSECS	Input	Security level, asserted to indicate a Non-secure transfer. For more information, see the *Arm® AMBA® 5 AHB Protocol Specification*.
HREADYS	Input	Data phase that is associated with the previous transfer on the interconnect is complete. The interconnect sends the signal to all AHB slaves and to the master, which started the transfer.
HREADYOUTS	Output	Slave ready.
HRDATAS[63:0]	Output	Read data.
HRESPS	Output	Slave response.
SAHBWABORT	Output	Indicates asynchronous abort for writes from TCM errors indicated on ITCMERR, D0TCMERR, D1TCMERR, D2TCMERR, or D3TCMERR.

C.10.1 S-AHB interface protection signals

The following table shows the *Slave AHB* (S-AHB) interface protection signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-12: S-AHB interface protection signals

Signal name	Direction	Description
HREADYCHKS	Input	Odd parity of **HREADYS**.
HREADYOUTCHKS	Output	Odd parity of **HREADYOUTS**.
HTRANSCHKS	Input	Odd parity of **HTRANSS[1:0]**.
HADDRCHKS[3:0]	Input	Odd parity of **HADDRS[31:0]** at 8-bit granularity.
HRDATACHKS[7:0]	Output	Odd parity of **HRDATAS[63:0]** at 8-bit granularity.
HWDATACHKS[7:0]	Input	Odd parity of **HWDATA[63:0]** at 8-bit granularity.
HWSTRBCHKS	Input	Odd parity of **HWSTRBS[7:0]**.
HPROTCHKS	Input	Odd parity of **HPROTS[6:0]**.
HCTRLCHK1S	Input	Odd parity of (**HBURSTS[2:0]**, **HNONSECS**, **HWRITES**, **HSIZES[2:0]**)
HRESPCHKS	Output	Odd parity of **HRESPS**.
HSELCHKS	Input	Odd parity of **HSELS**.
SAHBWABORTCHK	Output	Odd parity of **SAHBWABORT**.

C.11 P-AHB interface signals

The *Peripheral AHB* (P-AHB) interface implements the standard set of AMBA® 5 AHB signals.

The following table shows the signals for the P-AHB interface.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table C-13: P-AHB interface signals

Signal name	Direction	Description
HTRANSP[1:0]	Output	Transfer type.
HBURSTP[2:0]	Output	Transfer burst length.
HADDRP[31:0]	Output	Transfer address.
HWRITEP	Output	Write transfer.
HSIZEP[2:0]	Output	Transfer size.
HWDATAP[31:0]	Output	Write data.
HPROTP[6:0]	Output	Protection and outer memory attributes. **Note:** **HPROTP[0]** is always `0b1` as the interface does not support instruction fetch.
HNONSECP	Output	Asserted to indicate a Non-secure transfer.
HREADYP	Input	Slave ready.
HRDATAP[31:0]	Input	Read data.
HRESPP	Input	Slave response.
HMASTERP	Output	Initiator of the access: **0** Processor access. **1** Debugger access.
HEXCLP	Output	Exclusive request. Address phase control signal that indicates whether an access is a result of either a: • `LDREX` instruction. • `STREX` instruction. **0** Non-exclusive (standard) transaction. **1** Exclusive transaction.
HEXOKAYP	Input	Exclusive response. This data phase signal is sampled on **HREADYC**, and it indicates whether the exclusive request was granted. **0** Exclusive access failed. **1** Exclusive access that is granted.

C.11.1 P-AHB interface protection signals

The following table shows the *Peripheral AHB* (P-AHB) interface protection signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table C-14: P-AHB interface protection signals

Signal name	Direction	Description
HREADYCHKP	Input	Odd parity of **HREADYP**.
HTRANSCHKP	Output	Odd parity of **HTRANSP[1:0]**.
HADDRCHKP[3:0]	Output	Odd parity of **HADDRP[31:0]** at 8-bit granularity.
HRDATACHKP[3:0]	Input	Odd parity of **HRDATAP[31:0]** at 8-bit granularity.
HWDATACHKP[3:0]	Output	Odd parity of **HWDATA[31:0]** at 8-bit granularity.
HCTRLCHK1P	Output	Odd parity of (**HBURSTP[2:0]**, **HNONSECP**, **HWRITEP**, **HSIZEP[2:0]**)
HCTRLCHK2P	Output	Odd parity of (**HEXCLP**, **HMASTERP**)
HPROTCHKP	Output	Odd parity of **HPROTS[6:0]**.
HRESPCHKP	Input	Odd parity of (**HRESPP**, **HEXOKAYP**)

C.12 D-AHB interface signals

The following table shows the *Debug AHB* (D-AHB) interface signals.

Table C-15: D-AHB interface signals

Signal name	Direction	Description
HTRANSD[1:0]	Input	Indicates the type of current transfer. **Note:** **HTRANSD[0]** is ignored by the processor, all transactions are treated as either Non-sequential or Idle.
HBURSTD[2:0]	Input	Transfer burst length. Indicates whether the transfer is part of a burst. Debug accesses are always treated as SINGLE, and this signal is ignored.
HADDRD[31:0]	Input	Transfer address.
HWRITED	Input	Write transfer.
HSIZED[2:0]	Input	Transfer size. Indicates the size of the access. Accesses can be: `0b000` Byte. `0b001` Halfword. `0b010` Word. **Note:** **HSIZED[2]** is ignored by the processor.
HWDATAD[31:0]	Input	Write data. Data write bus.
HPROTD[6:0]	Input	Protection and outer memory attributes. Provides information on the access. **Note:** **HPROTD[0]** is ignored by the processor, all debug transactions are treated as data accesses.
HNONSECD	Input	Security level that is requested by debug access, asserted to indicate a Non-secure transfer. The resultant security level of the debug access depends on the debug control registers in the processor and the debug access control signals.
HREADYD	Output	Slave ready. When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.
HRDATAD[31:0]	Output	Read data.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
HRESPD	Output	Slave response

C.12.1 D-AHB interface protection signals

The following table shows the *Debug AHB* (D-AHB) interface signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-16: D-AHB interface protection signals

Signal name	Direction	Description
HREADYCHKD	Output	Odd parity of **HREADYD**.
HTRANSCHKD	Input	Odd parity of **HTRANSD[1:0]**.
HADDRCHKD[3:0]	Input	Odd parity of **HADDRD[31:0]** at 8-bit granularity.
HRDATACHKD[3:0]	Output	Odd parity of **HRDATAD[31:0]** at 8-bit granularity.
HWDATACHKD[3:0]	Input	Odd parity of **HWDATAD[31:0]** at 8-bit granularity.
HCTRLCHK1D	Input	Odd parity of (**HBURSTD[2:0]**, **HNONSECD**, **HWRITED**, **HSIZED[2:0]**).
HPROTCHKD	Input	Odd parity of **HPROTD[6:0]**.
HRESPCHKD	Output	Odd parity of **HRESPD**.

C.13 Debug EPPB interface signals

Debug *External Private Peripheral Bus* (EPPB) is an external peripheral interface. The following table shows the Debug EPPB APB interface signals.

Table C-17: Debug EPPB signals

Signal name	Direction	Description
PSEL	Output	APB device select. Indicates that a data transfer is requested.
PENABLE	Output	APB control signal. Strobe to time all accesses. Indicates the access phase of an APB transfer.
PPROT[2:0]	Output	Transfer privilege and security level.
PWRITE	Output	Write transfer.
PSTRB[3:0]	Output	Write data byte strobes
PADDR[19:2]	Output	Transfer address.
PADDR31	Output	Initiator of the transfer. **0** Processor **1** Debugger
PWDATA[31:0]	Output	APB 32-bit write data bus.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
PREADY	Input	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PSLVERR	Input	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.
PRDATA[31:0]	Input	APB 32-bit read data bus.

C.13.1 Debug EPPB interface protection signals

Debug *External Private Peripheral Bus* (EPPB) is an external peripheral interface. The following table shows the Debug EPPB interface protection signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-18: Debug EPPB interface protection signals

Signal name	Direction	Description
PSELCHK	Output	Odd parity of **PSEL**.
PREADYCHK	Input	Odd parity of **PREADY**.
PENABLECHK	Output	Odd parity of **PENABLE**.
PADDRCHK[3:0]	Output	Odd parity, at 8-bit granularity, of (**PADDR31**, `0b00000000000`,**PADDR[19:2]** ,`0b00`)
PRDATACHK[3:0]	Input	Odd parity of **PRDATA[31:0]** at 8-bit granularity.
PWDATACHK[3:0]	Output	Odd parity of **PWDATA[31:0]** at 8-bit granularity.
PCTRLCHK	Output	Odd parity of (**PPROT[2:0]**,**PWRITE**)
PSTRBCHK	Output	Odd parity of **PSTRB[3:0]**.
PSLVERRCHK	Input	Odd parity of **PSLVERR**.

C.14 Core EPPB interface signals

Core *External Private Peripheral Bus* (EPPB) is an external peripheral interface. The following table shows the Core EPPB APB interface signals.

Table C-19: Core EPPB signals

Signal name	Direction	Description
COREPSEL	Output	APB device select. Indicates that a data transfer is requested.
COREPENABLE	Output	APB control signal. Strobe to time all accesses. Indicates the access phase of an APB transfer.
COREPPROT[2:0]	Output	Transfer privilege and security level
COREPWRITE	Output	Write transfer
COREPSTRB[3:0]	Output	Write data byte strobes

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
COREPADDR[19:2]	Output	Transfer address
COREPADDR31	Output	Initiator of the transfer **0** Processor **1** Debugger
COREPWDATA[31:0]	Output	APB 32-bit write data bus
COREPREADY	Input	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
COREPSLVERR	Input	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.
COREPRDATA[31:0]	Input	APB 32-bit read data bus

Note

Arm recommends that all non-debug peripherals are integrated on the *Peripheral AHB* (P-AHB) interface.

C.14.1 Core EPPB interface protection signals

Core *External Private Peripheral Bus* (EPPB) is an external peripheral interface. The following table shows the Core EPPB interface protection signals.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-20: Core EPPB interface protection signals

Signal name	Direction	Description
COREPSELCHK	Output	Odd parity of **PSEL**.
COREPREADYCHK	Input	Odd parity of **PREADY**.
COREPENABLECHK	Output	Odd parity of **PENABLE**.
COREPADDRCHK[3:0]	Output	Odd parity, at 8-bit granularity, of (**PADDR31**,`0b00000000000`,**PADDR[19:2]** ,`0b00`)
COREPRDATACHK[3:0]	Input	Odd parity of **PRDATA[31:0]** at 8-bit granularity.
COREPWDATACHK[3:0]	Output	Odd parity of **PWDATA[31:0]** at 8-bit granularity.
COREPCTRLCHK	Output	Odd parity of (**PPROT[2:0]**,**PWRITE**)
COREPSTRBCHK	Output	Odd parity of **PSTRB[3:0]**.
COREPSLVERRCHK	Input	Odd parity of **PSLVERR**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

C.15 External coprocessor interface signals

The following table lists the external coprocessor interface signals.

Table C-21: External coprocessor interface signals

Signal name	Direction	Description
CPRESETOUTn	Output	This signal is asserted when the processor PDCORE domain is in reset.
CPENABLED[7:0]	Output	Indicates which coprocessor is enabled in the: • CPACR register associated with the Security state of the processor. • NSACR register if the processor is executing in Non-secure state. **Note:** The CPACR is banked.
CPPWRSU[7:0]	Output	Indicates which coprocessors are permitted to become **UNKNOWN**.
CPSPRESENT[7:0]	Input	Indicates which Secure coprocessors are present in the system.
CPNSPRESENT[7:0]	Input	Indicates which Non-secure coprocessors are present in the system.
CPCDP	Output	Coprocessor command operation.
CPMCR	Output	Coprocessor register transfer from processor operation.
CPMRC	Output	Coprocessor register transfer to processor operation.
CPSIZE	Output	Coprocessor size operation.
CPNUM[2:0]	Output	Coprocessor number request.
CPREGS[11:0]	Output	Operation register fields.
CPOPC[8:0]	Output	Operation opcode fields.
CPPRIV	Output	Indicates operation privilege.
CPNSATTR	Output	Indicates operation Security state.
CPFLUSH	Output	Indicates operations speculatively asserted valid in a prior phase will not assert valid.
CPSVALID	Output	Indicates that the speculative coprocessor operation is advancing to the second phase.
CPVALID	Output	Indicates whether the coprocessor operation is in the second phase.
CPREADY	Input	Indicates whether the coprocessor is stalled and used in the second phase.
CPERROR	Input	Indicates that the coprocessor is not present or the instruction is in the second cycle.
CPWDATA[63:0]	Output	The coprocessor write data bus.
CPRDATA[63:0]	Input	The coprocessor read data bus.

C.16 Debug interface signals

The following table shows the debug interface signals.

Note

For more information on debug authentication, see the section on authentication rules in the *Arm® CoreSight™ Architecture Specification v3.0*.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table C-22: Debug signals

Signal name	Direction	Description
HALTED	Output	In halting mode debug. **HALTED** remains asserted while the processor is in debug.
DBGRESTART	Input	Request for synchronized exit from halt mode. Forms a handshake with **DBGRESTARTED**. If multiprocessor debug support is not required, **DBGRESTART** must be tied LOW.
DBGRESTARTED	Output	Handshake for **DBGRESTART**.
EDBGRQ	Input	External debug request. A debug agent in the system asserts this signal to request that the processor enters Debug state.
DBGEN	Input	Invasive debug enable. When LOW, disables all halt-mode and invasive debug features.
NIDEN	Input	Non-invasive debug enable. When LOW, disables all trace and non-invasive debug features.
SPIDEN	Input	Secure invasive debug enable. When LOW, disables all halt mode and invasive debug features when the processor is in Secure state.
SPNIDEN	Input	Secure non-invasive debug enable. Controls access to non-invasive debug features when the processor is in Secure state and **SPIDEN** is LOW.

C.17 P-Channel and Q-Channel power control signals

The Cortex®-M85 processor PDCORE, PDEPU, and PDRAMS power domains are controlled by a P-Channel interface because there are multiple power modes, and each power mode is a combination of states for these domains. The debug power domain, PDDEBUG, is controlled by a Q-Channel interface because there are only two power modes, that is, ON and OFF.

PDCORE P-Channel interface signals

The following table shows the PDCORE, PDEPU, and PDRAMS P-Channel signals.

Note

- For more information on **COREPACTIVE** signal encoding, see 7.5.1 COREPACTIVE signal encoding on page 125.

Table C-23: PDCORE P-Channel interface signals

Signal name	Direction	Description
COREPREQ	Input	Request to transition to power mode indicated by **COREPSTATE**.
COREPSTATE[4:0]	Input	Requested power mode.
COREPACCEPT	Output	Acceptance of the transition to the requested power mode.
COREPDENY	Output	Denial of the power mode transition request.
COREPACTIVE[20:0]	Output	Active state or request for a power mode transition.
PWRDBGWAKEQACTIVE	Output	When there is a debugger (DAP) request to any EPPB address spaces in Cortex®-M85 MCU or external Debug EPPB interface or when there is a Core request to EPPB address spaces in PDDEBUG, the PDCORE domain asserts this signal. For more information, see 7.13 PWRDBGWAKEQACTIVE on page 130.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

PDDEBUG Q-Channel interface signals

The following table shows the PDDEBUG Q-Channel interface signals.

Note

The Q-Channel input **PWRDBGQREQn** signal is asynchronous to **DBGCLK** and is synchronized inside the Cortex®-M85 processor.

Table C-24: PDDEBUG Q-Channel interface signals

Signal name	Direction	Description
PWRDBGQREQn	Input	Debug domain quiescence request signal.
PWRDBGQACCEPTn	Output	Debug domain quiescence request accepted.
PWRDBGQDENY	Output	Debug domain quiescence request denied.
PWRDBGQACTIVE	Output	Debug logic active or activation request.
PWRCOREWAKEPACTIVE	Output	When there is a debugger (DAP) request to any address spaces in PDCORE, the PDDEBUG domain asserts this signal.

C.18 Q-Channel clock control signals

The **CLKIN** and **DBGCLK**, which can be gated at the system-level, is controlled by a separate Q-Channel interface.

The following table shows the Q-Channel signals for **CLKIN** clock control.

Note

The Q-Channel input **CLKINQREQn** signal is asynchronous to **CLKIN** and is synchronized inside the Cortex®-M85 processor.

Table C-25: Q-Channel for CLKIN control

Signal name	Direction	Description
CLKINQREQn	Input	Q-Channel for **CLKIN** control.
CLKINQACCEPTn	Output	
CLKINQDENY	Output	
CLKINQACTIVE	Output	

The following table shows the debug Q-Channel signals for **DBGCLK** clock control.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

The Q-Channel input **DBGCLKQREQn** signal is asynchronous to **DBGCLK** and is synchronized inside the Cortex®-M85 processor.

Table C-26: Q-Channel signals for DBGCLK control

Signal name	Direction	Description
DBGCLKQREQn	Input	Q-Channel for **DBGCLK** clock control.
DBGCLKQACCEPTn	Output	
DBGCLKQDENY	Output	
DBGCLKQACTIVE	Output	

C.19 Power compatibility control signals

The following table shows the power compatibility control signals.

Table C-27: Power compatibility control signals

Signal name	Direction	Description
SLEEPING	Output	When HIGH indicates that the processor is ready to enter a low-power state. When LOW, indicates that the processor is running or wants to leave sleep mode. If **SLEEPHOLDACKn** is LOW, then the processor does not perform any fetches until **SLEEPHOLDREQn** is driven HIGH.
SLEEPDEEP	Output	Indicates that the processor and ETM are ready to enter a low-power state and the wake up time is not critical. Only active when **SLEEPING** is HIGH.
SLEEPHOLDACKn	Output	Acknowledge signal for **SLEEPHOLDREQn**. If this signal is LOW, irrespective of the **SLEEPING** signal value, the processor does not advance in execution and does not perform any memory operations.
SLEEPHOLDREQn	Input	Request to extend the processor sleeping state regardless of wake up events. If the processor acknowledges this request driving **SLEEPHOLDACKn** LOW, this guarantees the processor remains idle even when receiving a wake up event.

C.20 ITM interface signals

The following table shows the ATB *Instrumentation Trace Macrocell* (ITM) interface signals.

Table C-28: ITM interface signals

Signal name	Direction	Description
AFREADYI	Output	Trace flush acknowledge.
AFVALIDI	Input	Trace flush request.
ATDATAI[7:0]	Output	Trace data.
ATIDI[6:0]	Output	Trace source ID.
ATREADYI	Input	Trace slave ready.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
ATVALIDI	Output	Trace transfer valid.
SYNCREQI	Input	ITM trace synchronization request.

C.21 ETM interface signals

The following table shows the ATB CoreSight™ *Embedded Trace Macrocell* (ETM) trace interface signals.

Table C-29: ETM interface signals

Signal name	Direction	Description
ATVALIDE	Output	Trace transfer is valid.
ATIDE[6:0]	Output	Trace source ID.
ATDATAE[7:0]	Output	Trace data.
AFREADYE	Output	Trace flush acknowledge.
AFVALIDE	Input	Trace flush request.
ATREADYE	Input	Trace slave is ready.
SYNCREQE	Input	ETM Trace synchronization request.

C.22 Trace synchronization and trigger signals

The following table shows the trace synchronization and trigger interface signals

Table C-30: Trace synchronization and trigger signals

Name	Type	Description
TRCENA	Output	Status of the DEMCR.TRCENA register, indicating whether the *Data Watchpoint Trace* (DWT) and *Instrumentation Trace Macrocell* (ITM) units are enabled (when implemented).
TPIUACTV	Input	TPIU data active.
TPIUBAUD	Input	TPIU Baud indicator
DSYNC	Output	DWT synchronization request.
ETMTRIGOUT	Output	ETM trigger event output bit[0]. Indicates a trigger packet in the trace stream.

C.23 CTI interface signals

The following table shows the *Cross Trigger Interface* (CTI) interface signals.

Table C-31: CTI signals

Signal name	Direction	Description
CTICHIN[3:0]	In	CTI channel input

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
CTICHOUT[3:0]	Out	CTI channel output
CTIIRQ[1:0]	Out	CTI interrupt request

C.24 Interrupt signals

All interrupt inputs must be generated synchronously to **CLKIN**. Both pulse and level interrupts are supported.

The following table shows the interrupt signals

Table C-32: Interrupt signals

Signal name	Direction	Description
IRQ[479:0]	Input	External interrupt signals. The `NUMIRQ` parameter configures the implemented bits of this signal. **Note:** • **IRQ** and **NMI** signals are active-HIGH and the hardware is agnostic between pulse- and level-signaled interrupts. • You must ensure that the **IRQ** and **NMI** signals to the processor are synchronized to **CLKIN** using the appropriate circuit.
NMI	Input	Non-maskable interrupt
CURRPRI[7:0]	Output	Current interrupt priority level. If the processor is in Handler mode for an exception with configurable priority **CURRPRI** indicates the programmed priority level of the exception. If the processor is in handler mode for an exception with negative priority **CURRPRI** is 0. If the processor is in Thread mode **CURRPRI** is dependent on whether a base priority mask is enabled by setting BASEPRI > 0: **BASEPRI==0** CURRPRI=0 **BASEPRI > 0** CURRPRI=BASEPRI The current exception number can be determined using the output signal **INTNUM**.
INTNUM[8:0]	Output	Interrupt number of the current execution context, from bits [8:0] of IPSR. **Note:** • When the processor is in Thread mode, **INTNUM** is 0. • When the processor is in Handler mode, **INTNUM** is the exception number of the currently executing exception.

C.25 WIC interface signals

There are two *Wakeup Interrupt Controller* (WIC) units that the processor supports.

- The *Internal Wakeup Interrupt Controller* (IWIC) that is present inside the processor.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- The *External Wakeup Interrupt Controller* (EWIC) that is an external peripheral to the processor.

WIC configuration signal

The following table shows the WIC configuration signal.

Table C-33: WIC configuration signal

Signal name	Direction	Description
WICCONTROL[3:0]	Input	This signal is responsible for WIC control and configuration. **WICCONTROL[3]** This bit indicates the EWIC automatic sequence on powerdown sequence. This bit is connected to EWIC in the system. **WICCONTROL[2]** This bit indicates the EWIC automatic sequence on powerup sequence. This bit is connected to EWIC in the system. **WICCONTROL[1]** This bit indicates that IWIC must be used. **WICCONTROL[0]** This bit indicates that **SLEEPDEEP** is WIC sleep. **Note:** • If No Wakeup interrupt controller is included, **WICCONTROL[3:0]** must be tied to `0b0000`. • If the IWIC is not included in the processor configuration, **WICCONTROL[1]** must be tied to `0b0`. If the EWIC is not included in the system, **WICCONTROL[3:1]**must be tied to `0b001`.

IWIC interface signals

The following table shows the IWIC signals.

Table C-34: IWIC signals

Signal name	Direction	Description
IWICCLK	Input	This signal is the IWIC clock.
nIWICRESET	Input	This is an active-LOW IWIC reset signal. This signal is internally synchronized to **IWICCLK**. The **nIWICRESET** signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronised inside the processor.
IWAKEUP	Output	This signal indicates the IWIC wake-up event that is detected when the processor is in WIC sleep.
IWICSENSE[482:0]	Output	This signal indicates which input events cause the WIC to generate the **IWAKEUP** signal. The `WICLINES` configuration parameter determines the usable width of this signal. Therefore, only the **IWICSENSE[WICLINES-1:0]** bits are implemented and the remaining bits are driven LOW. The mapping to input events is: **IWICSENSE[482:3]** **IRQ[479:0]**. **IWICSENSE[2]** **EDBGRQ**. **IWICSENSE[1]** **NMI**. **IWICSENSE[0]** **RXEV**. **Note:** • If No Wakeup interrupt controller is included, **WICCONTROL[3:0]** must be tied to `0b0000`. • If the IWIC is not included in the processor configuration, **WICCONTROL[1]** must be tied to `0b0`. If the EWIC is not included in the system, **WICCONTROL[3:1]**must be tied to `0b001`.

EWIC interface signal

The following table shows the EWIC signal.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Table C-35: EWIC signal

Signal name	Direction	Description
EWAKEUP	Input	The processor uses this signal to drive the **COREPACTIVE** output signal. This signal is asserted to indicate when a wakeup event is detected in WIC sleep. For more information on **COREPACTIVE**, see C.17 P-Channel and Q-Channel power control signals on page 361.

C.26 Event signals

The following table shows the event signals.

Table C-36: Event signals

Signal name	Direction	Description
TXEV	Output	This signal is a notification of an event that the processor generates when the `SEV` instruction is executed. This signal is a single-cycle pulse signal.
RXEV	Input	This signal is a notification of a system event. The processor expects a single cycle pulse signal.
LOCKUP	Output	This signal is a notification that the processor is in the architected lockup state because of an unrecoverable exception.
EVENTBUS[248:0]	Output	This signal indicates the *Performance Monitoring Unit* (PMU) events, which are intended to be used for system profiling. **EVENTBUS[n]** is pulsed for a single cycle for each event, n, on the processor. For a list of the PMU events, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*. By default, the **EVENTBUS** signal is only active when DWT, ITM, PMU, or ETM trace is enabled. If required, the **EVENTBUS** can be activated by setting ACTLR.EVENTBUSEN.

C.27 IDAU interface signals

An *Implementation Defined Attribution Unit* (IDAU) can control the security attributes for most of the memory the Cortex®-M85 processor addresses to a granularity of 32 bytes.

The following table shows the IDAU interface signals.

Table C-37: IDAU interface signals

Signal Name	Direction	Description
IDAUVALIDA	Output	Port A address valid.
IDAUADDRA[26:0]	Output	Port A address.
IDAUVALIDB	Output	Port B address valid.
IDAUADDRB[26:0]	Output	Port B address.
IDAUVALIDC	Output	Port C address valid.
IDAUADDRC[26:0]	Output	Port C address.
IDAUVALIDD	Output	Port D address valid.
IDAUADDRD[26:0]	Output	Port D address.
IDAUVALIDE	Output	Port E address valid.
IDAUADDRE[26:0]	Output	Port E address.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal Name	Direction	Description
IDAUNSA	Input	Port A Non-secure. This signal defines the attributes of IDAU region A.
IDAUNSCA	Input	Port A Non-secure Callable. This signal defines the attributes of IDAU region A.
IDAUNSB	Input	Port B Non-secure. This signal defines the attributes of IDAU region B.
IDAUNSCB	Input	Port B Non-secure Callable. This signal defines the attributes of IDAU region B.
IDAUNSC	Input	Port C Non-secure. This signal defines the attributes of IDAU region C.
IDAUNSCC	Input	Port C Non-secure Callable. This signal defines the attributes of IDAU region C.
IDAUNSD	Input	Port D Non-secure. This signal defines the attributes of IDAU region D.
IDAUNSCD	Input	Port D Non-secure Callable. This signal defines the attributes of IDAU region D.
IDAUNSE	Input	Port E Non-secure. This signal defines the attributes of IDAU region E.
IDAUNSCE	Input	Port E Non-secure Callable. This signal defines the attributes of IDAU region E.
IDAUIDA[7:0]	Input	Port A region number.
IDAUIDB[7:0]	Input	Port B region number.
IDAUIDC[7:0]	Input	Port C region number.
IDAUIDD[7:0]	Input	Port D region number.
IDAUIDE[7:0]	Input	Port E region number.
IDAUIDVA	Input	Port A region number valid.
IDAUIDVB	Input	Port B region number valid.
IDAUIDVC	Input	Port C region number valid.
IDAUIDVD	Input	Port D region number valid.
IDAUIDVE	Input	Port E region number valid.
IDAUNCHKA	Input	Port A region exempt from attribution check.
IDAUNCHKB	Input	Port B region exempt from attribution check.
IDAUNCHKC	Input	Port C region exempt from attribution check.
IDAUNCHKD	Input	Port D region exempt from attribution check.
IDAUNCHKE	Input	Port E region exempt from attribution check.

C.28 Miscellaneous signals

The following table shows the miscellaneous signals. The configuration input signals are sampled at reset.

Table C-38: Miscellaneous interface signals

Signal name	Direction	Description
TSVALUEB[63:0]	Input	Binary coded global timestamp count. This signal is synchronous to **CLKIN**.
TSCLKCHANGE	Input	This signal indicates timestamp clock ratio change.
SYSRESETREQ	Output	Request for functional reset. This can be done using either **nSYSRESET** or a combination of the P-Channel interface and **nSYSRESET**.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
COREECOREVNUM[7:0]	Input	ECO revision number for Core. The ECO revision field mappings are: **[7:4]** *System Control Space* (SCS) **[3:0]** CPUID This signal is expected to be static.
DBGECOREVNUM[27:0]	Input	ECO revision number for Debug. The ECO revision field mappings are: **[27:24]** *Performance Monitoring Unit* (PMU) **[23:20]** *Embedded Trace Macrocell* (ETM) **[19:16]** *Cross Trigger Interface* (CTI). **[15:12]** ROM table **[11:8]** *Instrumentation Trace Macrocell* (ITM) **[7:4]** *Data Watchpoint and Trace* (DWT) **[3:0]** *BreakPoint Unit* (BPU) This signal is expected to be static.
REVIDRNUM[3:0]	Input	Revision ID Number. The value of this signal is reflected in the Revision ID register REVIDR. This signal is expected to be static.
LOCKSVTAIRCR	Input	Disables writes to the following secure registers from software or from a debug agent that is connected to the processor. • VTOR_S. • AIRCR.PRIS. • AIRCR.BFHFNMINS. Asserting this signal: • Prevents changes to the secure vector table base address. • Handling of secure interrupt priority. • Handling of BusFault, HardFault, and NMI security target settings in the processor. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*. This signal can be changed dynamically.
LOCKNSVTOR	Input	Disables writes to the VTOR_NS register. For more information on this register, see *Arm®v8-M Architecture Reference Manual*. Asserting this signal prevents changes to the Non-secure vector table base address. This signal can be changed dynamically.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
LOCKSMPU	Input	This signal disables writes to registers that are associated with the Secure *Memory Protection Unit* (MPU) region from software or from a debug agent connected to the processor. • MPU_CTRL. • MPU_RNR. • MPU_RBAR. • MPU_RLAR. • MPU_RBAR_An. • MPU_RLAR_An. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*. Asserting this signal prevents changes to the memory regions which have been programmed in the secure MPU. All writes to the registers are ignored. This signal has no effect if no Secure MPU regions have been configured. This signal can be changed dynamically.
LOCKNSMPU	Input	This signal disables writes to registers that are associated with the Non-secure MPU region from software or from a debug agent connected to the processor. • MPU_CTRL_NS. • MPU_RNR_NS. • MPU_RBAR_NS. • MPU_RLAR_NS. • MPU_RBAR_A_NSn. • MPU_RLAR_A_NSn. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*. Asserting this signal prevents changes to the memory regions which have been programmed in the Non-secure MPU. All writes to the registers are ignored. This signal has no effect if the Cortex®-M85 processor has not been configured with support for Non-secure MPU regions. This signal can be changed dynamically.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
LOCKSAU	Input	This signal disables writes to registers that are associated with the *Security Attribution Unit* (SAU) region from software or from a debug agent connected to the processor. • SAU_CTRL. • SAU_RNR. • SAU_RBAR. • SAU_RLAR. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*. Asserting this signal prevents changes to the memory regions which have been programmed in the SAU. All writes to the registers are ignored. This signal has no effect if no SAU regions have been configured. This signal can be changed dynamically.
LOCKTCM	Input	This signal disables writes to registers that are associated with the TCM region from software or from a debug agent connected to the processor. • ITCMCR. • DTCMCR. For more information on these registers, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*. Asserting this signal prevents changes to the TCM configuration. All writes to the registers are ignored.
LOCKITGU	Input	This signal disables writes to registers that are associated with the ITCM interface security gating from software or from a debug agent connected to the processor. • ITGUCTRL. • ITGU_LUTn. For more information on these registers, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*. Asserting this signal prevents changes to the security gating configuration of the ITCM.
LOCKDTGU	Input	This signal disables writes to registers that are associated with the DTCM interface security gating from software or from a debug agent connected to the processor. • DTGUCTRL. • DTGU_LUTn. For more information on these registers, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*. Asserting this signal prevents changes to the security gating configuration of the DTCM.
LOCKPAHB	Input	Disable writes to the PAHBCR register from software or from a debug agent connected to the processor. For more information on this register, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*. Asserting this signal prevents changes to P-AHB port enable status in PAHBCR.EN.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
LOCKDCAIC	Input	Disable access to the instruction cache direct cache access registers DCAICLR and DCAICRR. Asserting this signal prevents direct access to the instruction cache Tag or Data RAM content. This is required when using *eXecutable Only Memory* (XOM) on the AXI master interface. When LOCKDCAIC is asserted: • DCAICLR is RAZ/WI. • DCAICRR is RAZ. For more information on these registers, see the *Arm® Cortex®-M85 Processor Technical Reference Manual*.

Note

- For more information on the ITCMCR and DTCMCR registers, see 5.19 ITCMCR and DTCMCR, TCM Control Registers on page 97.
- For more information on the ITGUCTRL and ITGU_LUTn registers, see 5.20 TCM security gate registers on page 98.
- For more information on DTGUCTRL and DTGU_LUTn registers, see 5.20 TCM security gate registers on page 98.
- For more information on the PAHBCR register, see 5.14 PAHBCR, P-AHB Control Register on page 86.
- For more information on the DCAICLR and DCAICRR registers, see 5.11.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers on page 72 and 5.11.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers on page 74.

C.29 Error interface signals

The error interface reports *Error Correcting Code* (ECC) errors that are detected in the caches and TCMs. The processor can report the location of up to two errors which occur simultaneously. It can also indicate if more than two errors have occurred, but cannot provide any additional information. The following table shows the error interface signals.

Table C-39: Error interface signals

Signal name	Direction	Description
DMEV0	Output	This signal indicates that an error is detected. When this signal is asserted, **DMEL0** and **DMEI0[25:0]** are valid.
DMEV1	Output	This signal indicates that at least two errors are detected. When this signal is asserted, **DMEL1** and **DMEI1[25:0]** are valid.
DMEV2	Output	This signal indicates that at least three errors are detected. No information about errors beyond the first two is sent.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
DMEL0[2:0]	Output	Location of the highest priority error detected. This is a one-hot signal and the format is: **DMEL0[2]** Error is found in the instruction cache. **DMEL0[1]** Error found in the data cache. **DMEL0[0]** Error found in the TCM.
DMEL1[2:0]	Output	Location of the second highest priority error detected. This is a one-hot signal and the format is: **DMEL1[2]** Error is found in the instruction cache. **DMEL1[1]** Error found in the data cache. **DMEL1[0]** Error found in the TCM.
DMEI0[25:0]	Output	Information about the highest priority error detected. This format of the signal depends on the location of the error: **Instruction cache** **DMEI0[14:0]** is the same format as bits [16:2] in IEBR0. **Data cache** **DMEI0[15:0]** is the same format as bits [17:2] in DEBR0. **TCM** **DMEI0[25:0]** is the same format as bits [27:2] in TEBR0. Unused bits of this signal are zero.
DMEI1[25:0]	Output	Information about the second highest priority error detected. This format of the signal depends on the location of the error: **Instruction cache** **DMEI1[14:0]** is the same format as bits [16:2] in IEBR1. **Data cache** **DMEI1[15:0]** is the same format as bits [17:2] in DEBR1. **TCM** **DMEI1[25:0]** is the same format as bits [27:2] in TEBR1. Unused bits of this signal are zero.
DBE[6:0]	Output	Detected Bus Error. A parity error has been detected from a protected interface. **Bit [6]** *Debug External Private Peripheral Bus* (Debug EPPB) parity error **Bit [5]** *Debug AHB* (D-AHB) APB parity error **Bit [4]** Reserved **Bit [3]** *Master AXI* (M-AXI) parity error **Bit [2]** *System AHB* (S-AHB) parity error **Bit [1]** *Peripheral AHB* (P-AHB) parity error **Bit [0]** *Core External Private Peripheral Bus* (Core EPPB) parity error A single-cycle pulse on the associated bit of **DBE** signals an error. This signal is always `0b0000000`.

C.30 Floating-point exception signals

The following table shows the floating-point exception signals.

The floating-point exception signals indicate mathematical errors that cause floating-point exceptions. Using these to indicate floating-point exceptions permits such exceptions to be diagnosed independently from software. For example, in safety-critical systems, exceptions can be routed directly to an on-chip safety controller.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

The floating-point exception signals are not related to the exception handling model. This means you can connect the floating-point exception signals to IRQ lines as your system design requires.

Table C-40: Floating-point signals

Signal name	Direction	Description
FPIXC	Output	Masked floating-point inexact exception
FPIDC	Output	Masked floating-point input denormal exception
FPOFC	Output	Masked floating-point overflow exception
FPUFC	Output	Masked floating-point underflow exception
FPDZC	Output	Masked floating-point divide-by-zero exception
FPIOC	Output	Invalid operation

C.31 PMC-100 interface signals

The following table shows the signals that are used only by the *Programmable MBIST Controller* (PMC-100). This interface contains control and configuration signals and PMC-100 APB signals used by an external agent to program the PMC-100. If the PMC-100 is not included in your processor configuration, then these signals are not used and must be tied off.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-41: PMC-100 control and configuration signals

Signal name	Direction	Description
PMCTEN	Input	Test enable. This is the master hardware enable for PMC-100. When this signal is asserted, on-line MBIST transactions can occur. When this signal is deasserted and tied LOW: • Only CoreSight™ registers and the internal PMC-100 control register are visible to reads from the memory mapped area in the PPB region. All other locations return zero. • All writes to the memory-mapped area in the PPB region are ignored. • On-line MBIST transactions do not occur. **Note:** This signal is sampled only at reset.
PMCTC	Input	Test continue pulse. This is a single cycle pulse and when enabled by an internal register in PMC-100, it causes a suspended test to continue execution.
PMCTE	Output	Test ended. When enabled in an internal register in PMC-100, this signal indicates that the test program has completed.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
PMCTF	Output	Test failed. When enabled in an internal register in the PMC-100, this signal indicates that a memory fault has been detected. **Note:** **PMCTF** and **PMCTE** may be asserted at the same time.

Table C-42: PMC-100 APB signals

Signal name	Direction	Description
PMCPSEL	Input	Transfer request
PMCPENABLE	Input	Indicates the second and subsequent cycles of an APB transfer
PMCPPROT[2:0]	Input	Transfer privilege and security level
PMCPWRITE	Input	Write transfer
PMCPADDR[11:2]	Input	Transfer address
PMCPWDATA[31:0]	Input	Write Data
PMCPSTRB[3:0]	Input	Write data byte strobes
PMCPREADY	Output	Slave ready
PMCPSLVERR	Output	Slave error response
PMCPRDATA[31:0]	Output	Read data
PMCPSELCHK	Input	Odd parity of **PSEL**
PMCPREADYCHK	Output	Odd parity of **PREADY**
PMCPENABLECHK	Input	Odd parity of **PENABLE**
PMCPADDRCHK[1:0]	Input	Odd parity of {4'b0000, PADDR[11:2], 2'b00} at 8-bit granularity
PMCPRDATACHK[3:0]	Output	Odd parity of **PRDATA[31:0]** at 8-bit granularity
PMCPWDATACHK[3:0]	Input	Odd parity of **PWDATA[31:0]** at 8-bit granularity
PMCPCTRLCHK	Input	Odd parity of {**PPROT[2:0]**, **PWRITE**}
PMCPSTRBCHK	Input	Odd parity of **PSTRB[3:0]**
PMCPSLVERRCHK	Output	Odd parity of **PSLVERR**

C.32 Test interface signals

The following tables show the *Design for Test* and production *Memory Built-In Self-Test* (MBIST) interface signals.

Table C-43: DFT signals

Signal name	Direction	Description
DFTCGEN	Input	Enables architectural clock gate override.
DFTRSTDISABLE[1:0]	Input	Disables synchronized multi-layer logic resets during scan shift. **DFTRSTDISABLE[0]** Disables the first level reset logic. **DFTRSTDISABLE[1]** Disables the second level reset logic.
DFTRAMHOLD	Input	Disable writes to the RAMs during scan shift.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Signal name	Direction	Description
nMBISTRESET	Input	Production MBIST reset.

Table C-44: Production MBIST interface signals

Signal name	Direction	Description
MBISTREQ	Input	Production MBIST mode request. **0** Normal operation **1** Production MBIST mode

C.33 DCLS operation signals

The following table shows the signals that the processor uses when configured with *Dual-Core Lock-Step* (DCLS) operation.

Note

These signals are reserved for future use in the Cortex®-M85 processor. These signals are not currently used and must be tied off.

Table C-45: DCLS operation signals

Signal name	Direction	Description
CLKINDCLS	Input	Signal for the redundant processor logic. This signal must be synchronous to **CLKIN**.
IWICCLKDCLS	Input	Signal for the redundant *Internal Wakeup Interrupt Controller* (IWIC) logic. This signal must be synchronous to **IWICCLK**.
nPORESETDCLS	Input	Power-on reset for the redundant processor logic. This signal must be asserted and de-asserted together with **nPORESET** . If **CLKIN** is active when **nPORESETDCLS** is asserted or de-asserted, then the signal must be constrained such that **nPORESETDLCS** is stable on the rising edge of the clock.
nSYSRESETDCLS	Input	System reset for the redundant processor logic. This signal must be asserted and de-asserted together with **nSYSRESET**. If **CLKIN** is active when **nSYSRESETDCLS** is asserted or de-asserted, then the signal must be constrained such that **nSYSRESETDLCS** is stable on the rising edge of the clock
nIWICRESETDCLS	Input	Power-on reset for the redundant IWIC logic. This signal must be asserted and de-asserted together with **nIWICRESET**. If **IWICCLK**is active when **nIWICRESETDCLS**is asserted or de-asserted, then the signal must be constrained such that **nIWICRESETDCLS** is stable on the rising edge of the clock.
DCLSCORECTL[11:0]	Input	Core DCLS feature control.
DCLSIWICCTL[5:0]	Input	IWIC DCLS feature control.
DCLSCORECOMPRES[11:0]	Output	Core DCLS comparator results.
DCLSIWICCOMPRES[5:0]	Output	IWIC DCLS comparator results.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Appendix D UNPREDICTABLE Behaviors

This appendix summarizes the behavior of the Cortex®-M85 processor in cases where the Arm®v8.1-M architecture is **UNPREDICTABLE**.

D.1 Use of instructions defined in architecture variants

An instruction that is provided by one or more of the architecture extensions is either **UNPREDICTABLE** or **UNDEFINED** in an implementation that does not include those extensions.

In the Cortex®-M85 processor, all instructions that are not explicitly supported generate an UNDEFINSTR UsageFault exception.

D.2 Use of Program Counter - R15 encoding

R15 is **UNPREDICTABLE** as a source or destination in most data processing operations. R15 is also **UNPREDICTABLE** as a transfer register in certain load/store instructions. Examples of such instructions include LDRT, LDRH, and LDRB.

In the Cortex®-M85 processor, the use of R15 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

D.3 Use of Stack Pointer - as a general-purpose register R13

R13 is defined in the Thumb instruction set so that its use is primarily as a stack pointer. R13 is normally identified as stack pointer, SP in Thumb instructions.

In 32-bit Thumb instructions, if you use SP as a general-purpose register beyond the architecturally defined constraints, the results are **UNPREDICTABLE**.

In the Cortex®-M85 processor, the use of R13 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

In the architecture where the use of R13 as a general-purpose register is defined, bits[1:0] of the register must be treated as SBZP. Writing a nonzero value to bits [1:0] results in **UNPREDICTABLE** behavior. In the Cortex®-M85 processor, bits [1:0] of R13 are always RAZ/WI.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

D.4 Register list in load and store multiple instructions

Load and Store Multiple instructions (`LDM`, `STM`, `PUSH`, `POP`, `VLDM`, and `VSTM`) transfer multiple registers to and from consecutive memory locations using an address from a base register, which can be optionally written back when the operation is complete.

The registers are selected from a list encoded in the instruction. Some of these encodings are **UNPREDICTABLE**.

In the Cortex®-M85 processor:

- R13 is ignored in any instruction encodings which include R13 in the register list.
- For a Load Multiple, if PC is specified in the list and the instruction is in an IT block and is not the final instruction, a fault is not generated. The branch is taken and the IT state is cleared.
- For a Store Multiple instruction, storing of the PC is ignored.
- For a Load Multiple instruction, if base writeback is specified and the register to be written back is also in the list to be loaded, the instruction performs all the loads in the specified addressing mode and the register being written back takes the loaded value.
- For a Store Multiple instruction, if base writeback is specified and the register to be written back is also the first register in the list to be stored, the value stored is the initial base register value. The base register is written back with the expected updated value. If the register to be written back is not the first register in the list, then it takes the updated value.
- For a floating-point Load or Store Multiple instruction, `VLDM`, `VSTM`, `VPUSH`, and `VPOP`, if the register list extends beyond S63 or D31, then the Cortex®-M85 processor ignores all registers that are greater than S31 or D15. If it has a writeback, then the base register becomes **UNDEFINED**.

D.5 Exception-continuable instructions

To improve interrupt response and increase processing throughput, the processor can take an interrupt during the execution of a Load Multiple or Store Multiple instruction, and continue execution of the instruction after returning from the interrupt. During the interrupt processing, the EPSR.ICI bit holds the continuation state of the Load Multiple or Store Multiple instruction.

In the Cortex®-M85 processor, if an exception-continuable instruction is interrupted, then modification of the EPSR.ICI bits by either the software or a debugger might generate an INVSTATE UsageFault exception when re-execution of the interrupted instruction is attempted.

This includes the architecturally **UNPREDICTABLE** cases of:

- Not a register in the register list of the Load Multiple or Store Multiple instruction.
- The first register in the register list of the Load Multiple or Store Multiple instruction.

The Cortex®-M85 processor also generates an INVSTATE UsageFault exception if the ICI bits are set to any non-zero value for an integer Load Multiple instruction with the base register in the

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

register list, and ICI set to a greater register number than the base register. This is because these instructions are not eligible for continuation.

D.6 Stack limit checking

The Arm®v8.1-M architecture defines the instructions which are subject to stack limit checking when operating on SP.

It states that it is **UNKNOWN** whether a stack limit check is performed on any use of the SP that was **UNPREDICTABLE** in Arm®v7-M and Arm®v6-M. In the Cortex®-M85 processor, these **UNPREDICTABLE** cases are when R13 is used as a general-purpose register in instructions. In these circumstances, the processor generates an UNDEFINSTR UsageFault exception.

D.7 UNPREDICTABLE instructions within an IT block

Instructions executed in an IT block which change the PC are architecturally **UNPREDICTABLE** unless they are the last instruction in the block.

In the Cortex®-M85 processor:

- The instructions below are treated as UNDEFINED, but the UNDEFINSTR UsageFault will only be generated if the applied IT condition passes:
  - Conditional branch instructions (`Bcond label`)
  - Compare and Branch instruction, `CBNZ` and `CBZ`
  - An `IT` instruction inside another IT block
  - `CPS` instructions
- The instructions below execute normally when they are not the last instructions in the IT block:
  - Unconditional branch instructions (`B label`)
  - Branch with link instructions (`BL label`) - note that the processor treats `BLX` PC as **UNPREDICTABLE** and will raise an UNDEFINSTR UsageFault, but only if the applied IT condition passes
  - Branch and exchange instructions (`BX Rm`)
  - Table branch instructions (`TBB` and `TBH`)
- If the Floating-Point Extension is included and one of the instructions below is executed in an IT block, these instructions are treated as UNDEFINED, but the UNDEFINSTR UsageFault will only be generated if the applied IT condition passes:
  - `VCVTA`
  - `VCVTN`
  - `VCVTP`
  - `VCVTM`

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - VMAXNM
    - VMINNM
    - VRINTA
    - VRINTN
    - VRINTP
    - VRINTM
    - VSEL
- A branch or `PC` writing instruction that is not the last in an IT block will always clear the ITSTATE.
- If the PACBTI Extension is included, the BTI and PACBTI instructions execute unconditionally inside an IT block.

D.8 Memory access and address space

In the Arm®v8.1-M architecture, there are memory accesses that result in **UNPREDICTABLE** behavior in the Cortex®-M85 processor.

The following table shows the memory accesses that are **UNPREDICTABLE** and the Cortex®-M85 processor behavior.

Table D-1: Memory accesses and Cortex®-M85 processor behavior

Memory access	Cortex®-M85 processor behavior
Any access to memory from a load or store instruction or an instruction fetch, which overflows the 32-bit address space.	These kinds of accesses wrap around to addresses at the start of memory.
For any access X, the bytes accessed by X must all have the same memory type attribute, otherwise the behavior of the access is **UNPREDICTABLE**. That is, an unaligned access that spans a boundary between different memory types is **UNPREDICTABLE**.	In the Cortex®-M85 processor, each part of an access to a different 32-byte aligned region is dealt with independently. Each access to a different 32-byte region makes a new MPU lookup.
For any two memory accesses X and Y that are generated by the same instruction, the bytes accessed by X and Y must all have the same memory type attribute. Otherwise, the results are **UNPREDICTABLE**. For example, an `LDC`, `LDM`, `LDRD`, `STC`, `STM`, `STRD`, `VSTM`, `VLDM`, `VPUSH`, `VPOP`, `VLDR`, or `VSTR` that spans a boundary between Normal and Device memory is **UNPREDICTABLE**.	In the Cortex®-M85 processor, each part of access to a different 32-byte aligned region is dealt with independently. Each access to a different 32-byte aligned region makes a new MPU lookup.
Any instruction fetch must only access Normal memory. If it accesses Device memory, the result is **UNPREDICTABLE**. For example, instruction fetches must not be performed to an area of memory that contains read-sensitive devices because there is no ordering requirement between instruction fetches and explicit accesses.	In the Cortex®-M85 processor, fetches to Device memory are sent out to the system, indicated on the M-AXI interface as Device, unless the memory region is marked with the *Execute Never* (XN) memory attribute.
The behavior of sequential instruction fetches that cross from Non-secure to Secure memory and fulfill the secure entry criteria specified in the architecture, including the presence of a *Secure Gateway* (`SG`) instruction at the boundary of the secure memory area, is CONSTRAINED **UNPREDICTABLE**.	In the Cortex®-M85 processor, this results in a fault (INVEP).

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Memory access	Cortex®-M85 processor behavior
A load to the *Program Counter* (PC) from a non-word-aligned address is **UNPREDICTABLE**.	In the Cortex®-M85 processor, this results in an alignment fault.

D.9 MPU programming

The Arm *Protected Memory System Architecture* (PMSA) includes many **UNPREDICTABLE** cases when programming the MPU

In the Cortex®-M85 processor:

- Setting MPU_CTRL.ENABLE to 0 and MPU_CTRL.HFNMIEA to 1 is **UNPREDICTABLE**. This results in all memory accesses using the default memory map including those from Exception Handlers with a priority less than one.
- If MPU_RNR is written with a region number greater than the number of regions defined in the MPU, then the value used is masked by one less than the number of regions defined. For example:
  - The number of regions defined is given as num_regions. The value written to MPU_RNR is given as v.
  - num_regions=8 and v=9.
  - The effective region used is given as 9 & (8-1); region 1.

  The number of regions available can be read from MPU_TYPE.DREGION.
- Setting MPU_RBAR.SH to 1 is **UNPREDICTABLE**. This encoding is treated as Non-shareable.
- The Attribute fields (MPU_ATTR) of the MPU_MAIR0 and MPU_MAIR1 registers include some encodings which are **UNPREDICTABLE**.
  - If MPU_ATTR[7:4]!=0 and MPU_ATTR[3:0]==0 is **UNPREDICTABLE**, the attributes are treated as Normal memory, Outer non-cacheable, Inner non-cacheable.
  - If MPU_ATTR[7:4]==0 and MPU_ATTR[1:0]!=0 is **UNPREDICTABLE**, the attributes are treated as Device-nGnRE.
- The external AMBA® 5 AHB interface signals cannot distinguish between some of the memory attribute encodings defined by the PMSA:
  - Normal transient memory is treated the same as Normal non-transient memory.
  - Device memory with Gathering or Reordering attributes (G, R) are always treated as non-Gathering and non-Reordering. Early Write Acknowledgment attributes (E, nE) are supported on the Cortex®-M85 AHB5 interfaces.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

D.10 Miscellaneous UNPREDICTABLE instruction behavior

This section documents the behavior of the Cortex®-M85 processor in a number of miscellaneous **UNPREDICTABLE** instruction scenarios:

- Load instructions that specify writeback of the base register are **UNDEFINED** if the base register to be written back matches the register to be loaded (Rn==Rt). These cases generate an **UNDEFINSTR** UsageFault exception.
- Store instructions that specify writeback of the base register are **UNDEFINED** if the base register to be written back matches the register to be stored (Rn==Rt). The base register is then written back with the expected updated value. These cases generate an **UNDEFINSTR** UsageFault exception.
- Multiply and Multiply accumulate instructions that write a 64-bit result using two registers, `SMULL`, `SMLAL`, `SMLALBB`, `SMLALBT`, `SMLALTB`, `SMLALTT`, `SMLALD`, `SMLALDX`, `SMLSLD`, `SMLSLDX`, `UMULL`, and `UMAAL` are **UNPREDICTABLE** if the two registers are the same (RdHi==RdLo). In the Cortex®-M85 processor, these cases generate an **UNDEFINSTR** UsageFault exception.
- Floating-point instructions that transfer between two registers and either two single-precision registers or one double-precision register, `VMOV Rt, Rt2, Dm` and `VMOV Rt, Rt2, Sm, Sm1` are **UNPREDICTABLE** if the two registers are the same (Rt==Rt2). In the Cortex®-M85 processor, these cases generate an **UNDEFINSTR** UsageFault exception.

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Appendix E Revisions

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

E.1 Revisions

The following tables show any significant technical changes between released issues of this book.

Table E-1: Issue 0000-02

Change	Location
First beta release for r0p0	-

Table E-2: Differences between issue 0000-02 and issue 0000-03

Change	Location
First limited access release for r0p0	-
Editorial changes	Throughout document
Progressive terminology commmitment renamed to Inclusive language commitment and editorial content updates made	Document front matter
Information about TPIU-M documentation added	1.1.4 Additional reading on page 20
Figure for Example processor system replaced	2.1 Cortex-M85 processor overview on page 22
Details for PACBTI updated in table of processor architectural features	2.2 Cortex-M85 features on page 23
Information about LOW OVERHEAD BRANCH (LOB) feature removed in bullet point that mentions instruction queue	2.3 Supported standards and specifications on page 25
Entry for Debug and Core EXTERNAL PRIVATE PERIPHERAL BUS (EPPB) interfaces split into separate rows in the Interfaces table	3.2 Interfaces on page 37
APB entry replaced with PMC-100 external interface	3.2 Interfaces on page 37
New topic added	4.6.2 Multicycle instructions on page 48
Updates to the System control register summary table: • REVIDR added • CFSR description updated • ID_DFR0 note under reset value updated • ID_ISAR0 reset value explanation updated • CCSIDR reset value updated	5.1 System control register summary on page 49
Updates to the Identification register summary table: • ID_DFR0 note under reset value updated • CTR reset value updated • CCSIDR reset value updated • REVIDR new note added under reset value	5.2 Identification register summary on page 52
CTR reset value updated	5.5 Cache identification register summary on page 59
Topic title updated	5.5.3 CCSIDR, Current Cache Size ID Register on page 62

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Note about REVIDR[3:0] added	5.6 REVIDR, Revision ID Register on page 63
Updates to the **IMPLEMENTATION DEFINED** registers summary: • TEBRDATA0 updated to reflect not being accessible from software • TEBRDATA1 updated to reflect not being accessible from software • DCADCRR added missing reset value • DCAICRR added missing reset value	5.10 IMPLEMENTATION DEFINED registers summary on page 68
Updates to the Error bank registers table: • TEBRDATA0 updated to reflect not being accessible from software • TEBRDATA1 updated to reflect not being accessible from software	5.12 Error bank registers on page 78
Information updated to reflect that these registers are not accessible from software	5.12.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1 on page 83
Bit assignment information updated	5.15 PFCR, Prefetcher Control Register on page 88
Updates to Configuration paramteter selection table All mentions of the `IRQTIER` parameter removed	5.17.1 CFGINFOSEL, Processor configuration information selection register on page 92
All mentions of the `IRQTIER` parameter removed	5.17.2 CFGINFORD, Processor configuration information read data register on page 95
Updates to the TCM security gate registers table: • ITGU_CFG reset value updated • DTGU_CFG reset value updated	5.20 TCM security gate registers on page 98
Field 0 in bit assingments updated	5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers on page 105
Erroneously duplicated topic removed	6.6 Enabling branch prediction on page 111
Erroneously removed topic restored	6.7 Enabling the branch cache on page 111
Last column for COREPACTIVE[8:0] most significant set bit updated and the row for MEM_RET (No cache) removed in the bits encodings table Note under the bits encodings table updated to reflect changes made in the table	7.5 COREPACTIVE and required power mode on page 123
DEEPSLEEP changed to SLEEPDEEP and clarification added to bullet about CPDLPSTATE.CLPSTATE	7.6 PDCORE low-power requirements on page 125
Bullet list updated	7.7 PDEPU low-power requirements on page 126
Additional inforamtion about CPDLPSTATE.RLPSTATE selecting the low-power state added Bullet item about automatic cache invalidation updated	7.8 PDRAMS powerdown requirements on page 127
Requirements for entering WARM_RST updated Information about asserting nSYSRESET updated Paragraph about debug access being made on D-AHB removed	7.9 Warm reset power mode on page 127
IPPB and EPPB tables updated	8.3 Private Peripheral Bus on page 134
Note added to the table with the examples	8.4 Unaligned accesses on page 136
Processor and TCM address ranges updated in the figure	9.8.1 Memory aliasing and IDAU/SAU configuration on page 147
Details for Read issuing capability updated in the attributes and transactions tables	10.4.1.1 High performance configuration M-AXI attributes and transactions on page 163
The MPU violation behavior sub-topic title expanded to include SAU and IDAU	10.3.1.1 Considerations for system design on page 160
New paragraph about stream detection added	10.4.1.2 Data prefetching on page 165

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Additional clarification added about accesses to Device memory	10.4.5 Memory system implications for AXI accesses on page 168
Bullet item about high-performance M-AXI configuration and outstanding accesses removed	10.4.6.1 Restrictions on AXI transfers on page 169
Information added on how to configure the processor to support ECC	• 10.8 TCM interfaces on page 178 • 10.8.1 TCM configuration on page 179
Bullet items for operations in Non-secure state updated	10.9.3 Cache maintenance operations on page 186
Error prioritization diagram updated to emphasize the source and type of error	11.2.3 Error detection and processing on page 200
Updates to the Parity checking conditions table: • Clarification of the parity checking conditions for M-AXI • SAHBWABORT added to the entry for S-AHB	Interface protection behavior
New note added to the description of NUM[15:0]	11.4.6 ERRDEVID, RAS Error Record Device ID Register on page 215
Updates to reset values table: • FPDSCR updated reset value when floating-point and MVE are not included. • FPSCR updated reset value when floating-point and MVE are not included.	14.3 FPDSCR and FPSCR register reset values on page 226
Note added about debugger cannot write to IPSR	15.1 Debug functionality on page 228
Updates made to the ROM table components table: • ITM reset value • TPIU row updates • Component ID registers upper address range value corrected Correction of PIDR0 reset value in processor ROM table identification values.	15.1.3 Processor ROM table identification and entries on page 231
Title updated to replace powerdown with reset New note added about D-AHB in PDEBUG Updates to the Debug and trace registers accessible during processor reset table: • System Control and ID registers group added • Memory address range updated for some entries New paragraph added after the Debug and trace regsters table	15.2.4 Debug during reset and before code execution commences on page 240
New note after PMU events table	16.2 PMU events on page 243
Reset value updated for register PMU_AUTHSTATUS	16.3 PMU register summary on page 249
Information about TPIU-M support added	B.1 TPIU features on page 313
• Note updated • IMPLEMENTATION DEFINED added to the title of the register summary table • TPIU_PIDR3 reset value and note updated	B.2 TPIU register summary on page 315
• DCLS information added to **nPORESET** and **NSYSRESET** • **nPORESETDCLS**, **nSYSRESETDCLS**, and **nWICRESETDCLS** signals removed	C.2 Reset signals on page 341
CFGMVE[1:0] description updated for clarity	C.3 Static configuration signals on page 342
Clarification to the note about **ITCMWR** and **ITCMBYTEWR** added	C.7 Instruction Tightly Coupled Memory interface signals on page 346

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Descriptions for **ARID[2:0]** and **RID[2:0]** updated	C.9 M-AXI interface signals on page 349
AWAKEUPCHK added	C.9.1 M-AXI interface protection signals on page 352
SAHBWABORTCHK added	C.10.1 S-AHB interface protection signals on page 354
First note under P-Channel interface signals replaced with new content Description for **PWRCOREWAKEPACTIVE** in PDDEBUG Q-Channel interface signals table updated to remove mention of IPPB and EPPB	C.17 P-Channel and Q-Channel power control signals on page 361
Description for **WICCONTROL[3:0]** updated	C.25 WIC interface signals on page 365
EVENTBUS width updated to [248:0]	C.26 Event signals on page 367
Updates to signals table: • **ECOREVNUM[35:0]** removed • **COREECOREVNUM[7:0]** added • **DBGECOREVNUM[27:0]** added • **REVIDRNUM[3:0]** added	C.28 Miscellaneous signals on page 368
Previous PMC-100 signals table split into two tables: control and configuration signals, and APB signals Update to control and configuration signals table: **PMCTF** new note in description added Update to APB signals table: Incorrect spelling for **PMCPENABLE** fixed	C.31 PMC-100 interface signals on page 374
Width and description for **DFTRSTDISABLE[1:0]** updated	C.32 Test interface signals on page 375
Updates to signals table: • **DCLSCTL[23:0]** removed • **DCLSCOMPRES[23:0]** removed • **DCLSCORECTL[11:0]** added • **DCLSIWICCTL[5:0]** added • **DCLSCORECOMPRES[11:0]** added • **DCLSIWICCOMPRES[5:0]** added	DCLS operation signals
New topic added	Control and reporting
Bullet point about PACBTI Extension added	D.7 UNPREDICTABLE instructions within an IT block on page 379
Clarification added to some bullet points	D.10 Miscellaneous UNPREDICTABLE instruction behavior on page 381

Table E-3: Differences between issue 0000-03 and issue 0001-04

Change	Location
Updated processor revision number to r0p1	-
First early access release for r0p1	-
Editorial changes	Throughout document

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Removed content associated with functional safety, including for functionality associated with: • **DUAL CORE LOCK-STEP** (DCL) • Interface protection (BUSPROT) • *Programmable MBIST Controller* (PMC-100) • **ON-LINE MBIST** interface • *Software Built-In Self-Test* (SBIST) controller • Software Test Library (STL) Signal tables applicable to any of the removed items have been retained, accompanied by a note indicating future availability and tie-off information. Figures that include components associated with Functional Safety include a note indicating future support. Otherwise, all other content removal includes all applicable inline contextual information, partial chapters, and full topics.	Throughout document
Topic updated for clarity and technical accuracy	3.1.1 Cortex-M85 processor core on page 31
MAU block diagram figure updated for clarity and technical accuracy	3.1.3.1 Memory Authentication Unit on page 33
Modifications to the processor configurable options table include: • Unsupported entry "Lowest interrupt latency interrupt numbers" removed • Entry for number of IRQ lines updated for clarity and technical accuracy	3.6 Cortex-M85 implementation options on page 39
CPUID register reset value updated for r0p1 revision	• 5.1 System control register summary on page 49 • 5.2 Identification register summary on page 52
Entry for ERRIIDR register added	5.2 Identification register summary on page 52
Number in Revision bit field updated to 0x1 for r0p1	5.4 CPUID, CPUID Base Register on page 58
PFCR register reset value updated	5.10 IMPLEMENTATION DEFINED registers summary on page 68
CFGINFOSEL register usage constraints updated for technical accuracy	5.17.1 CFGINFOSEL, Processor configuration information selection register on page 92
CFGINFORD register usage constraints updated for technical accuracy	5.17.2 CFGINFORD, Processor configuration information read data register on page 95
List of steps for operating mode transitioning updated for technical accuracy	7.3.1 Operating mode transitions which change PDRAMS power state on page 119
Note below table "COREPSTATE and COREPACTIVE bits encodings" updated for technical accuracy	7.5 COREPACTIVE and required power mode on page 123
Topic expanded and updated for technical accuracy	7.7 PDEPU low-power requirements on page 126
Number of IDAU external interfaces updated	9.4 Implementation Defined Attribution Unit on page 144
Prefectcher information updated for clarity and technical accuracy	10.4.1.2 Data prefetching on page 165
Topic updated for technical accuracy	10.7 EPPB interfaces on page 177
Topic expanded and updated for clarity and technical accuracy	10.10.2 Store buffer behavior on page 193
Topic updated for technical accuracy	11.2 ECC memory protection behavior on page 198

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Topic expanded and updated for technical accuracy	11.2.1 ECC schemes and error type terminology on page 199
DHCSR register reset value updated	15.1.5 Debug register summary on page 234
Subtopic "Unprivileged Debug is enabled " updated for technical accuracy	15.2.2 Debugger access memory attributes and data cache access on page 236
Function information for PSRCount bits updated and expanded for technical accuracy	B.2.4 TPIU_PSCR, Periodic Synchronization Counter Register on page 319
Description of **CFGITCMSZ[3:0]** signal updated to fix an erroneous encoding value calculation in ITCM size	C.3 Static configuration signals on page 342
Removed note that is most appropriate for the Arm® Cortex®-M85 Processor Integration and Implementation Manual	C.14.1 Core EPPB interface protection signals on page 359
Signal names for **CLKINQACCEPTn** and **PWRCOREWAKEPACTIVE** corrected in table "Q-Channel for CLKIN control"	C.18 Q-Channel clock control signals on page 362
Topic updated for technical accuracy	D.10 Miscellaneous UNPREDICTABLE instruction behavior on page 381

Table E-4: Differences between issue 0001-04 and issue 0002-05

Change	Location
First release for r0p2	-
Updated the product name to Cortex-M85	Throughout document
CPUID register reset value updated for r0p2 revision	• 5.1 System control register summary on page 49 • 5.2 Identification register summary on page 52
Updated the CCSIDR register description	• 5.1 System control register summary on page 49 • 5.5 Cache identification register summary on page 59
Updated the LOCKED bit function description	5.12.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1 on page 79
Updated function descriptions	5.12.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1 on page 80
Updated the condition related to CTI	7.10 Debug Q-Channel and PDDEBUG power domain on page 129
Moved CTI info to PDDEBUG	7.1 Power domains on page 115
Added a third case for the PDCORE domain	7.13 PWRDBGWAKEQACTIVE on page 130
Updated EPPB to Core EPPB and Debug EPPB	• 8.5 Access privilege level for Device and Normal memory on page 137 • 10.1 Memory system features on page 153
Removed a statement related to INCR N 64-bit	10.4.1 High performance M-AXI configuration on page 162
Removed Read-allocation under the configuration associative	10.4.1 High performance M-AXI configuration on page 162
Updated restrictions related to DSB and PLD	10.4.6.1 Restrictions on AXI transfers on page 169
Updated multiple bit descriptions	11.4.2 ERRSTATUS0, RAS Error Record Primary Status Register on page 209
Updated the reset value of PMU	15.1.3 Processor ROM table identification and entries on page 231
Topic updated for technical accuracy	15.2.4 Debug during reset and before code execution commences on page 240
Changed CTIINTACK to CTI_INTACK	19 Cross Trigger Interface on page 266

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Change	Location
Changed **CTICHOUT[3:0]** to **CTITRIGOUT[3:0]**	19.5 CTI_APPSET, CTI Application Channel Set Register on page 271
Updated field values for REL	19.29 CTI_PIDR2, Peripheral Identification Register 2 on page 292
Updated EPPB to their respective Core and Debug EPPBs	• A External Wakeup Interrupt Controller on page 300 • B.2.18 TPIU_PIDR4, Peripheral Identification Register 4 on page 331
Updated the device memory configuration information for M-AXI INCR sections	• 10.4.1.1 High performance configuration M-AXI attributes and transactions on page 163 • 10.4.2.1 Area optimized configuration M-AXI attributes and transactions on page 165

Copyright © 2020–2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential