Copyright Notice

Copyright © 2022 Andes Technology Corporation. All rights reserved.

AndesCore™, AndeSight™, AndeShape™, AndeSLive™, AndeSoft™, AndeStar™, Andes Custom Extension™, CoDense™, StackSafe™, QuickNap™, AndesClarity™, AndeSim™, AndeSysC™, Andes-E Embedded and Driving Innovations are trademarks owned by Andes Technology Corporation. All other trademarks used herein are the property of their respective owners.

The product described herein is subject to continuous development and improvement. Thus, all information herein is provided by Andes in good faith but without warranties. This document is intended only to assist the reader in the use of the product. Andes Technology Corporation shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

Contact Information

Should you have any problems with the information contained herein, you may contact Andes Technology Corporation through at support@andestech.com

Please include the following information in your inquiries:

• the document title
• the document number
• the page number(s) to which your comments apply
• a concise explanation of the problem.

General suggestions for improvements are welcome.
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Rev. Date</th>
<th>Revised Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>2022-03-25</td>
<td>Third Release for CPU Revision 5.0.0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Revise disclaimers on Copyright Notice page</td>
</tr>
<tr>
<td>1.1</td>
<td>2022-03-01</td>
<td>Second Release for CPU Revision 5.0.0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Revise wording of nceplmt100 and ncepldm200 in section 19.2.1 and 20.3.12 respectively</td>
</tr>
<tr>
<td>1.0</td>
<td>2021-12-17</td>
<td>First Release for CPU Revision 5.0.0.</td>
</tr>
</tbody>
</table>
# Contents

## Revision History

### List of Figures

### List of Tables

## 1 Overview

1.1 AX45MP-1C Processor Features ............................................................. 1
1.2 Block Diagram .................................................................................................. 4
1.3 Major Components ............................................................................................ 4
1.4 Pipeline Stages and Activities .......................................................................... 5

## 2 Instruction Set Overview

2.1 Introduction ........................................................................................................ 7
2.2 Integer Registers .................................................................................................. 7
2.3 Atomic Instructions ............................................................................................... 8
  2.3.1 Load-Reserved/Store-Conditional Instruction .............................................. 8
  2.3.2 Atomic Memory Operation Instruction ....................................................... 8
2.4 Misaligned Memory Access .................................................................................. 8
  2.4.1 Exceptions .................................................................................................... 9
2.5 Floating-Point ISA Extension ............................................................................ 9
  2.5.1 Support for Half-Precision and BFLOAT16 Formats .................................... 9
2.6 DSP ISA Extension ............................................................................................... 10

## 3 Branch Prediction Unit

## 4 Memory Management Unit

4.1 Introduction ........................................................................................................ 12
4.2 Address Translation ............................................................................................. 12
4.3 Translation Lookaside Buffer ............................................................................ 14
  4.3.1 Instruction uTLB (iTLB) ............................................................................ 14
  4.3.2 Data uTLB (dTLB) ................................................................................... 14
  4.3.3 Shared TLB (STLB) .................................................................................. 14
  4.3.4 Replacement Policy .................................................................................... 14
4.4 Page Table Walker (PTW) ............................................................................... 14
  4.4.1 Introduction ................................................................................................. 14
  4.4.2 Page Table Address Formation ................................................................. 15
4.5 Attributes for Address Spaces ............................................................................ 15
## Attributes for Virtual Memory Pages

- **Attributes for Virtual Memory Pages**

## Local Memory

### Introduction

5.1 Introduction

5.2 Local Memory Spaces

5.3 Local Memory Address Range

5.4 Local Memory Usage Constraints

## Local Memory Slave Port

### Introduction

6.1 Introduction

6.2 Support for Soft Error Protection

## Level-1 Caches

### Introduction

7.1 Introduction

7.2 Cache Replacement Policy

7.3 I-Cache

7.3.1 I-Cache Fill Operation

7.3.2 I-Cache Prefetch

7.3.3 I-Cache TAG SRAM Fields

7.4 D-Cache

7.4.1 Cache Access Latency

7.4.2 D-Cache Fill Operations

7.4.3 D-Cache Eviction Operations

7.4.4 D-Cache Write Buffers

7.4.5 D-Cache TAG SRAM Fields

7.5 FENCE/FENCE.I Operations

7.6 CCTL Operations

7.6.1 Invalidating Cache Blocks (L1D_VA_INVAL, L1I_VA_INVAL, L1D_IX_INVAL, L1I_IX_INVAL)

7.6.2 Writing Back Cache Blocks (L1D_VA_WB, L1D_IX_WB, L1D_WB_ALL)

7.6.3 Writing Back & Invalidating Cache Blocks (L1D_VA_WBINVAL, L1D_IX_WBINVAL, L1D_WBINVAL_ALL)

7.6.4 Filling and Locking Cache Blocks (L1D_VA_LOCK, L1I_VA_LOCK)

7.6.5 Unlocking Cache Blocks (L1D_VA_UNLOCK, L1I_VA_UNLOCK)

7.6.6 Reading Tag Data from Caches (L1D_IX_RTAG, L1I_IX_RTAG)

7.6.7 Reading Data from D-Cache (L1D_IX_RDATA)

7.6.8 Reading Data from I-Caches (L1I_IX_RDATA)

7.6.9 Writing Tag Data to Caches (L1D_IX_WTAG, L1I_IX_WTAG)

7.6.10 Writing Data to D-Cache (L1D_IX_WDATA)

7.6.11 Writing Data to I-Caches (L1I_IX_WDATA)
7.6.12 Invalidating All Cache Blocks (L1D_INVAL_ALL) ........................................... 31
7.6.13 Writing Back All Cache Blocks (L1D_WB_ALL) ........................................... 31
7.6.14 Writing Back & Invalidating All Cache Blocks (L1D_WBINVAL_ALL) .......... 31
7.7 Supervisor/User CCTL Operations .................................................................... 31
8 Level-2 Cache ....................................................................................................... 33
  8.1 Introduction ........................................................................................................ 33
    8.1.1 Multi-Cycle RAM Support ........................................................................... 33
    8.1.2 L2-Cache Prefetch ....................................................................................... 34
    8.1.3 Asynchronous Error ...................................................................................... 34
    8.1.4 Cache Control Operation ............................................................................. 34
  8.2 L2-Cache Organization ...................................................................................... 35
    8.2.1 Two Bank L2-Cache Structure .................................................................... 35
  8.3 L2C Programming Guide .................................................................................. 35
  8.4 L2-Cache Controller Registers ......................................................................... 36
    8.4.1 Register Type ............................................................................................... 36
    8.4.2 Summary of Registers .................................................................................. 36
    8.4.3 Configuration Register ................................................................................. 38
    8.4.4 Control Register ........................................................................................... 39
    8.4.5 HPM Control Register 0 .............................................................................. 42
    8.4.6 HPM Control Register 1 .............................................................................. 44
    8.4.7 HPM Control Register 2 .............................................................................. 45
    8.4.8 HPM Control Register 3 .............................................................................. 46
    8.4.9 Asynchronous Error Register ....................................................................... 47
    8.4.10 Error Register ............................................................................................. 48
    8.4.11 CCTL Command Registers 0 – 3 ................................................................. 50
    8.4.12 CCTL Access Line Registers 0 – 3 ............................................................... 51
    8.4.13 CCTL Status Register ............................................................................... 53
    8.4.14 CCTL TGT Data Registers 0 – 7 ................................................................. 54
    8.4.15 CCTL TGT ECC Code Register ................................................................. 56
    8.4.16 HPM Counter Registers 0 – 31 .................................................................. 57
9 L1 Memory Error Protection .................................................................................. 59
  9.1 Parity/ECC Control Mode .................................................................................. 59
    9.1.1 Disable Parity/ECC ...................................................................................... 60
    9.1.2 Generate Exceptions on Uncorrectable Parity/ECC Errors ......................... 60
    9.1.3 Generate Exceptions on Parity/ECC Errors ............................................... 61
  9.2 Local Memory Protection .................................................................................. 61
  9.3 I-Cache Protection ............................................................................................ 61
  9.4 D-Cache Protection ........................................................................................... 62
9.5 Soft Error Injection ................................. 62
  9.5.1 ILM/DLM ECC Error Injection ................. 62
  9.5.2 I-Cache Parity/ECC Error Injection ........... 62
  9.5.3 D-Cache ECC Error Injection .................. 63

10 Physical Memory Attributes ................. 64
  10.1 Introduction ..................................... 64
  10.2 Static Physical Memory Attributes .......... 64
  10.3 Programmable Physical Memory Attributes .... 64
  10.4 Memory Access Ordering ......................... 65
  10.5 Non-Cacheable Memory and Device Accesses . 65
  10.6 Cacheable Memory Accesses to L1-Caches .... 65
      10.6.1 Write-Back ................................. 66
      10.6.2 I-Cache Disabled Behaviors ............... 67
      10.6.3 D-Cache Disabled Behaviors ............... 67
      10.6.4 Debug Mode ................................ 67
  10.7 Cacheable Memory Accesses to L2-Caches .... 68

11 MemBoost ........................................... 70
  11.1 Non-Blocking Memory Access .................... 70
  11.2 D-Cache Write-Around Support ................ 71
  11.3 D-Cache Prefetch ................................ 72

12 Coherence Manager (CM) ....................... 74

13 Trap ................................................ 75
  13.1 Introduction ................................... 75
  13.2 Interrupt ....................................... 75
      13.2.1 Additional Local Interrupts .............. 76
      13.2.2 Interrupt Status and Masking ............ 76
  13.3 Exception ...................................... 76
  13.4 Trap Handling .................................. 77
      13.4.1 Entering the Trap Handler ................. 77
      13.4.2 Returning from the Trap Handler .......... 78

14 Reset and Non-Maskable Interrupts ........ 79
  14.1 Reset ........................................... 79
  14.2 Non-Maskable Interrupts ....................... 79

15 Power Management ............................... 80
  15.1 Power Management Unit ......................... 80
15.2 Wait-For-Interrupt Mode .................................................. 80
15.3 Dynamic Frequency Scaling (DFS) ........................................ 81
15.4 D-Cache Coherency .......................................................... 81
  15.4.1 Disable D-Cache Coherency ........................................... 81
  15.4.2 Enable D-Cache Coherency ........................................... 81
15.5 Disable All Clocks of a Core .............................................. 82

16 Control and Status Registers .............................................. 83
16.1 Introduction ................................................................... 83
  16.1.1 System Register Type .................................................. 83
  16.1.2 Reset Value .............................................................. 83
  16.1.3 CSR Listing .............................................................. 83
16.2 Machine Information Registers ......................................... 91
  16.2.1 Machine Vendor ID Register ........................................ 91
  16.2.2 Machine Architecture ID Register ................................ 91
  16.2.3 Machine Implementation ID Register ......................... 92
  16.2.4 Hart ID Register ....................................................... 93
16.3 Machine Trap Related CSRs ............................................... 93
  16.3.1 Machine Status ........................................................ 93
  16.3.2 Machine ISA Register ............................................... 98
  16.3.3 Machine Exception Delegation ..................................... 99
  16.3.4 Machine Interrupt Delegation ...................................... 102
  16.3.5 Machine Interrupt Enable .......................................... 104
  16.3.6 Machine Trap Vector Base Address .............................. 106
  16.3.7 Machine Scratch Register .......................................... 107
  16.3.8 Machine Exception Program Counter ............................ 107
  16.3.9 Machine Cause Register .......................................... 108
  16.3.10 Machine Trap Value ................................................ 110
  16.3.11 Machine Interrupt Pending ....................................... 111
  16.3.12 Machine Extended Status ......................................... 113
  16.3.13 Machine Detailed Trap Cause .................................... 114
  16.3.13.1 Detailed Exception Priority .................................. 117
  16.3.14 Machine Supervisor Local Interrupt Delegation ............ 117
16.4 Counter Related CSRs ..................................................... 119
  16.4.1 Machine Cycle Counter ............................................. 119
  16.4.2 Machine Instruction-Retired Counter ............................. 120
  16.4.3 Machine Performance Monitoring Counter ..................... 120
  16.4.4 Machine Counter-Inhibit ........................................... 120
  16.4.5 Machine Performance Monitoring Event Selector ............. 120
16.4.6 Machine Counter Enable ........................................ 125
16.4.7 Machine Counter Write Enable .............................. 126
16.4.8 Machine Counter Interrupt Enable ......................... 126
16.4.9 Machine Counter Mask for Machine Mode ................ 127
16.4.10 Machine Counter Mask for Supervisor Mode ............. 127
16.4.11 Machine Counter Mask for User Mode .................... 128
16.4.12 Machine Counter Overflow Status ........................ 128

16.5 Configuration Control & Status Registers ..................... 129
16.5.1 Instruction Cache/Memory Configuration Register .... 129
16.5.2 Data Cache/Memory Configuration Register .............. 133
16.5.3 Misc. Configuration Register ............................... 137
16.5.4 L2-Cache Control Base Register ......................... 142

16.6 Trigger Registers .................................................. 143
16.6.1 Trigger Select ............................................... 143
16.6.2 Trigger Data 1 ............................................... 143
16.6.3 Trigger Data 2 ............................................... 145
16.6.4 Trigger Data 3 ............................................... 146
16.6.5 Trigger Info ................................................. 147
16.6.6 Trigger Control ............................................. 149
16.6.7 Machine Context ............................................ 149
16.6.8 Supervisor Context ......................................... 150
16.6.9 Match Control .............................................. 150
16.6.10 Instruction Count ......................................... 153
16.6.11 Interrupt Trigger .......................................... 154
16.6.12 Exception Trigger ......................................... 156
16.6.13 Trigger Extra .............................................. 158

16.7 Debug and Trigger Registers .................................... 159
16.7.1 Debug Control and Status Register ....................... 159
16.7.2 Debug Program Counter .................................... 162
16.7.3 Debug Scratch Register 0 .................................. 163
16.7.4 Debug Scratch Register 1 .................................. 163
16.7.5 Exception Redirection Register ............................ 163
16.7.6 Debug Detailed Cause ..................................... 167

16.8 Supervisor Trap Related CSRs .................................. 170
16.8.1 Supervisor Status ........................................... 170
16.8.2 Supervisor Exception Delegation .......................... 173
16.8.3 Supervisor Interrupt Delegation ........................... 176
16.8.4 Supervisor Interrupt Enable ................................ 177
16.8.5 Supervisor Trap Vector Base Address .................... 179
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.8.6 Supervisor Counter Enable Register</td>
<td>180</td>
</tr>
<tr>
<td>16.8.7 Supervisor Scratch Register</td>
<td>181</td>
</tr>
<tr>
<td>16.8.8 Supervisor Exception Program Counter</td>
<td>182</td>
</tr>
<tr>
<td>16.8.9 Supervisor Cause Register</td>
<td>183</td>
</tr>
<tr>
<td>16.8.10 Supervisor Trap Value</td>
<td>185</td>
</tr>
<tr>
<td>16.8.11 Supervisor Interrupt Pending</td>
<td>186</td>
</tr>
<tr>
<td>16.8.12 Supervisor Local Interrupt Enable</td>
<td>188</td>
</tr>
<tr>
<td>16.8.13 Supervisor Local Interrupt Pending</td>
<td>190</td>
</tr>
<tr>
<td>16.8.14 Supervisor Detailed Trap Cause</td>
<td>192</td>
</tr>
<tr>
<td>16.8.14.1 Detailed Exception Priority</td>
<td>195</td>
</tr>
<tr>
<td>16.9 Supervisor Translation Related CSRs</td>
<td>196</td>
</tr>
<tr>
<td>16.9.1 Supervisor Address Translation and Protection</td>
<td>196</td>
</tr>
<tr>
<td>16.10 Supervisor Counter Related CSRs</td>
<td>197</td>
</tr>
<tr>
<td>16.10.1 Supervisor Counter Mask for Machine Mode</td>
<td>197</td>
</tr>
<tr>
<td>16.10.2 Supervisor Counter Mask for Supervisor Mode</td>
<td>198</td>
</tr>
<tr>
<td>16.10.3 Supervisor Counter Mask for User Mode</td>
<td>199</td>
</tr>
<tr>
<td>16.10.4 Supervisor Counter Interrupt Enable</td>
<td>200</td>
</tr>
<tr>
<td>16.10.5 Supervisor Counter Overflow Status</td>
<td>201</td>
</tr>
<tr>
<td>16.10.6 Supervisor Counter-Inhibit</td>
<td>202</td>
</tr>
<tr>
<td>16.10.7 Supervisor Performance Monitoring Event Selector</td>
<td>203</td>
</tr>
<tr>
<td>16.11 User Trap Related CSRs</td>
<td>204</td>
</tr>
<tr>
<td>16.11.1 User Status</td>
<td>204</td>
</tr>
<tr>
<td>16.11.2 User Interrupt Enable</td>
<td>205</td>
</tr>
<tr>
<td>16.11.3 User Trap Vector Base Address</td>
<td>206</td>
</tr>
<tr>
<td>16.11.4 User Scratch Register</td>
<td>207</td>
</tr>
<tr>
<td>16.11.5 User Exception Program Counter</td>
<td>208</td>
</tr>
<tr>
<td>16.11.6 User Cause Register</td>
<td>209</td>
</tr>
<tr>
<td>16.11.7 User Trap Value</td>
<td>211</td>
</tr>
<tr>
<td>16.11.8 User Interrupt Pending</td>
<td>212</td>
</tr>
<tr>
<td>16.11.9 User Detailed Trap Cause</td>
<td>213</td>
</tr>
<tr>
<td>16.12 Memory and Miscellaneous Registers</td>
<td>214</td>
</tr>
<tr>
<td>16.12.1 Instruction Local Memory Base Register</td>
<td>214</td>
</tr>
<tr>
<td>16.12.2 Data Local Memory Base Register</td>
<td>216</td>
</tr>
<tr>
<td>16.12.3 ECC Code Register</td>
<td>218</td>
</tr>
<tr>
<td>16.12.4 NMI Vector Base Address Register</td>
<td>220</td>
</tr>
<tr>
<td>16.12.5 Performance Throttling Control Register</td>
<td>221</td>
</tr>
<tr>
<td>16.12.6 Cache Control Register</td>
<td>222</td>
</tr>
<tr>
<td>16.12.7 Machine Miscellaneous Control Register</td>
<td>227</td>
</tr>
<tr>
<td>16.12.8 Clock Control Register</td>
<td>229</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>16.12.9</td>
<td>Machine CCTL Begin Address</td>
</tr>
<tr>
<td>16.12.10</td>
<td>Machine CCTL Command</td>
</tr>
<tr>
<td>16.12.11</td>
<td>Machine CCTL Data</td>
</tr>
<tr>
<td>16.12.12</td>
<td>Supervisor CCTL Data</td>
</tr>
<tr>
<td>16.12.13</td>
<td>User CCTL Begin Address</td>
</tr>
<tr>
<td>16.12.14</td>
<td>User CCTL Command</td>
</tr>
<tr>
<td>16.13</td>
<td>Hardware Stack Protection and Recording Registers</td>
</tr>
<tr>
<td>16.13.1</td>
<td>Machine Hardware Stack Protection Control</td>
</tr>
<tr>
<td>16.13.2</td>
<td>Machine SP Bound Register</td>
</tr>
<tr>
<td>16.13.3</td>
<td>Machine SP Base Register</td>
</tr>
<tr>
<td>16.14</td>
<td>CoDense Registers</td>
</tr>
<tr>
<td>16.14.1</td>
<td>Instruction Table Base Address Register</td>
</tr>
<tr>
<td>16.15</td>
<td>DSP Registers</td>
</tr>
<tr>
<td>16.15.1</td>
<td>Code Register</td>
</tr>
<tr>
<td>16.16</td>
<td>Physical Memory Protection Unit Configuration &amp; Address Registers</td>
</tr>
<tr>
<td>16.16.1</td>
<td>PMP Configuration Registers</td>
</tr>
<tr>
<td>16.16.2</td>
<td>PMP Address Register</td>
</tr>
<tr>
<td>16.17</td>
<td>Physical Memory Attribute Unit Configuration &amp; Address Registers</td>
</tr>
<tr>
<td>16.17.1</td>
<td>PMA Configuration Registers</td>
</tr>
<tr>
<td>16.17.2</td>
<td>PMA Address Register</td>
</tr>
<tr>
<td>16.18</td>
<td>Floating-Point CSRs</td>
</tr>
<tr>
<td>16.18.1</td>
<td>Floating-Point Accrued Exception Flags</td>
</tr>
<tr>
<td>16.18.2</td>
<td>Floating-Point Rounding Mode</td>
</tr>
<tr>
<td>16.18.3</td>
<td>Floating-Point Control and Status</td>
</tr>
<tr>
<td>16.19</td>
<td>User Counter Related CSRs</td>
</tr>
<tr>
<td>16.19.1</td>
<td>Cycle Counter</td>
</tr>
<tr>
<td>16.19.2</td>
<td>User Time Register</td>
</tr>
<tr>
<td>16.19.3</td>
<td>Instruction-Retired Counter</td>
</tr>
<tr>
<td>16.19.4</td>
<td>Performance Monitoring Counter</td>
</tr>
</tbody>
</table>

17 Instruction Throughput and Latency

17.1 ALU Instructions

17.2 Dual-Issue Capability

17.2.1 Dual-Issue Capability of Integer Instructions

17.3 Throughput and Latency for Aligned Load Instructions

17.4 Throughput and Latency for Misaligned Load Instructions

17.5 Divide and Remainder Instructions

17.6 Branch and Jump Instruction

17.7 EXEC.IT Instruction
17.8 CSR Instruction ......................................................... 266
  17.8.1 Latency Type ................................................... 266
  17.8.2 List of CSRs with Latency Type WH ....................... 267
  17.8.3 List of CSRs with Latency Type WA ....................... 268
  17.8.4 List of CSRs with Latency Type RF ....................... 268

17.9 Trap Return Instruction ........................................... 269

17.10 FENCE Instruction ................................................ 269

17.11 Scalar Floating-Point Instructions for CPU Revision 10.0.0 and Later ............................................... 269

17.12 DSP Instructions .................................................... 270

18 Platform-Level Interrupt Controller (PLIC) 271
  18.1 Introduction ....................................................... 271
  18.2 Support for Preemptive Priority Interrupt ..................... 272
    18.2.1 Interrupt Claims with Preemptive Priority ............... 273
    18.2.2 Interrupt Completion with Preemptive Priority ........... 273
    18.2.3 Programming Sequence to Allow Preemption of Interrupts .... 273
  18.3 Vectored Interrupts ............................................... 274
  18.4 PLIC Registers .................................................... 275
    18.4.1 Summary of Registers ....................................... 275
    18.4.2 Feature Enable Register .................................... 276
    18.4.3 Interrupt Source Priority .................................. 276
    18.4.4 Interrupt Pending ........................................... 277
    18.4.5 Interrupt Trigger Type ..................................... 277
    18.4.6 Number of Interrupt and Target Configuration Register .... 278
    18.4.7 Version & Maximum Priority Configuration Register ........ 278
    18.4.8 Interrupt Enable Bits for Target m ....................... 279
    18.4.9 Priority Threshold for Target m ......................... 279
    18.4.10 Claim and Complete Register for Target m ............... 280
    18.4.11 Preempted Priority Stack Registers for Target m ........ 280

19 Machine Timer 282
  19.1 Introduction ..................................................... 282
  19.2 Machine Timer Registers ....................................... 282
    19.2.1 Machine Timer Initialization ............................ 283

20 Debug Subsystem 284
  20.1 Overview ......................................................... 284
  20.2 Optional Debug Subsystem ....................................... 285
  20.3 DebugModule ...................................................... 286
    20.3.1 Abstract Data 0–3 (data0–data3) ........................ 288
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.3.2 Debug Module Control (dmcontrol)</td>
<td>288</td>
</tr>
<tr>
<td>20.3.3 Debug Module Status (dmstatus)</td>
<td>291</td>
</tr>
<tr>
<td>20.3.4 Hart Info (hartinfo)</td>
<td>293</td>
</tr>
<tr>
<td>20.3.5 Halt Summary (haltsum)</td>
<td>294</td>
</tr>
<tr>
<td>20.3.6 Hart Array Window Select (hawindowsel)</td>
<td>294</td>
</tr>
<tr>
<td>20.3.7 Hart Array Window (hawindow)</td>
<td>294</td>
</tr>
<tr>
<td>20.3.8 Abstract Control and Status (abstractcs)</td>
<td>295</td>
</tr>
<tr>
<td>20.3.9 Abstract Command (command)</td>
<td>296</td>
</tr>
<tr>
<td>20.3.9.1 Access Register</td>
<td>297</td>
</tr>
<tr>
<td>20.3.9.2 Quick Access</td>
<td>297</td>
</tr>
<tr>
<td>20.3.9.3 Access Memory</td>
<td>298</td>
</tr>
<tr>
<td>20.3.10 Abstract Command Autoexec (abstractauto)</td>
<td>299</td>
</tr>
<tr>
<td>20.3.11 Device Tree Addr 0–3 (devtreeaddr0 – devtreeaddr3)</td>
<td>299</td>
</tr>
<tr>
<td>20.3.12 Program Buffer 0–15 (progbuf0 – progbuf15)</td>
<td>299</td>
</tr>
<tr>
<td>20.3.13 Authentication Data (authdata)</td>
<td>299</td>
</tr>
<tr>
<td>20.3.14 System Bus Access Control and Status (sbcs)</td>
<td>300</td>
</tr>
<tr>
<td>20.3.15 System Bus Address (sbaddress0 – sbaddress2)</td>
<td>302</td>
</tr>
<tr>
<td>20.3.16 System Bus Data (sbdatal0 – sbdata3)</td>
<td>302</td>
</tr>
<tr>
<td>20.4 JTAG Debug Transport Module</td>
<td>302</td>
</tr>
<tr>
<td>20.4.1 BYPASS</td>
<td>303</td>
</tr>
<tr>
<td>20.4.2 IDCODE</td>
<td>303</td>
</tr>
<tr>
<td>20.4.3 DTM Control and Status (dtmcs)</td>
<td>304</td>
</tr>
<tr>
<td>20.4.4 Debug Module Interface Access (dmi)</td>
<td>305</td>
</tr>
</tbody>
</table>
List of Figures

1. AX45MP-1C Block Diagram ................................................................. 4
2. Virtual Address to Physical Address Translation ....................... 13
3. NCEPLIC100 Block Diagram ............................................................... 272
4. Debug Subsystem Block Diagram ..................................................... 284
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integer Registers</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Translated Address Space Attribute</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Priorities for Instruction Fetches</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Priorities for Data Accesses</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Local Memory Address Range (for ILM and DLM)</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Configuration Choices for the I-Cache</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>Configuration Choices for the D-Cache</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>I-Cache TAG SRAM Fields</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>Access Latency of the D-Cache</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>D-Cache TAG SRAM Fields</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Effects of FENCE/FENCE.I Instructions</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>Addressing Type of CCTL Commands</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Index Format for D-Cache Index Type of CCTL Operations</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Index Format for I-Cache Index Type of CCTL Operations</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>User CCTL Operations</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>L2C Setup Cycle Control</td>
<td>33</td>
</tr>
<tr>
<td>17</td>
<td>L2C Data Output Cycle Control</td>
<td>33</td>
</tr>
<tr>
<td>18</td>
<td>Supported L2C CCTL Operations</td>
<td>35</td>
</tr>
<tr>
<td>19</td>
<td>L2C Register Summary</td>
<td>36</td>
</tr>
<tr>
<td>20</td>
<td>Monitored Event Definitions of the L2C Performance Counter</td>
<td>42</td>
</tr>
<tr>
<td>21</td>
<td>L2 Tag Data Width</td>
<td>54</td>
</tr>
<tr>
<td>22</td>
<td>Memory Protection Types</td>
<td>59</td>
</tr>
<tr>
<td>23</td>
<td>Memory Access Ordering</td>
<td>65</td>
</tr>
<tr>
<td>24</td>
<td>Behaviors upon Accessing Write-Back and No-Allocate Regions (MTYP=8)</td>
<td>66</td>
</tr>
<tr>
<td>25</td>
<td>Behaviors upon Accessing Write-Back and Read-Allocate Regions (MTYP=9)</td>
<td>66</td>
</tr>
<tr>
<td>26</td>
<td>Behaviors upon Accessing Write-Back and Write-Allocate Regions (MTYP=10)</td>
<td>66</td>
</tr>
<tr>
<td>27</td>
<td>Behaviors upon Accessing Write-Back and Read-and-Write-Allocate Regions (MTYP=11)</td>
<td>67</td>
</tr>
<tr>
<td>28</td>
<td>Behaviors upon Accessing Write-Back and No-Allocate Regions (MTYP=8)</td>
<td>68</td>
</tr>
<tr>
<td>29</td>
<td>Behaviors upon Accessing Write-Back and Read-Allocate Regions (MTYP=9)</td>
<td>68</td>
</tr>
<tr>
<td>30</td>
<td>Behaviors upon Accessing Write-Back and Write-Allocate Regions (MTYP=10)</td>
<td>68</td>
</tr>
<tr>
<td>31</td>
<td>Behaviors upon Accessing Write-Back and Read-and-Write-Allocate Regions (MTYP=11)</td>
<td>69</td>
</tr>
<tr>
<td>32</td>
<td>Machine Information Registers</td>
<td>83</td>
</tr>
<tr>
<td>33</td>
<td>Machine Trap Related Registers</td>
<td>84</td>
</tr>
<tr>
<td>34</td>
<td>Counter Related Registers</td>
<td>84</td>
</tr>
<tr>
<td>35</td>
<td>Configuration Control &amp; Status Registers</td>
<td>85</td>
</tr>
<tr>
<td>36</td>
<td>Trigger Registers</td>
<td>85</td>
</tr>
<tr>
<td>37</td>
<td>Debug Registers</td>
<td>85</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>38</td>
<td>Supervisor Trap Related Registers</td>
<td>86</td>
</tr>
<tr>
<td>39</td>
<td>Supervisor Page Translation Related Registers</td>
<td>86</td>
</tr>
<tr>
<td>40</td>
<td>Supervisor Counter Related Registers</td>
<td>86</td>
</tr>
<tr>
<td>41</td>
<td>User Trap Related Registers</td>
<td>87</td>
</tr>
<tr>
<td>42</td>
<td>Memory and Miscellaneous Registers</td>
<td>87</td>
</tr>
<tr>
<td>43</td>
<td>Hardware Stack Protection and Recording Registers</td>
<td>88</td>
</tr>
<tr>
<td>44</td>
<td>CoDense Registers</td>
<td>88</td>
</tr>
<tr>
<td>45</td>
<td>DSP Registers</td>
<td>88</td>
</tr>
<tr>
<td>46</td>
<td>PMP Registers</td>
<td>88</td>
</tr>
<tr>
<td>47</td>
<td>PMA Registers</td>
<td>89</td>
</tr>
<tr>
<td>48</td>
<td>Floating-Point CSRs</td>
<td>89</td>
</tr>
<tr>
<td>49</td>
<td>User Mode Counter Related Registers</td>
<td>89</td>
</tr>
<tr>
<td>50</td>
<td>Possible Values of mcause After Trap</td>
<td>108</td>
</tr>
<tr>
<td>51</td>
<td>Possible Values of mcause After Reset</td>
<td>109</td>
</tr>
<tr>
<td>52</td>
<td>Possible Values of mcause After NMI</td>
<td>110</td>
</tr>
<tr>
<td>53</td>
<td>Possible Values of mcause After Vector Interrupt</td>
<td>110</td>
</tr>
<tr>
<td>54</td>
<td>Event Selectors</td>
<td>121</td>
</tr>
<tr>
<td>55</td>
<td>Virtual Address in DPC upon Debug Mode Entry</td>
<td>163</td>
</tr>
<tr>
<td>56</td>
<td>AX45MP-1C scause Value After Trap</td>
<td>183</td>
</tr>
<tr>
<td>57</td>
<td>AX45MP-1C ucause Value After Trap</td>
<td>209</td>
</tr>
<tr>
<td>58</td>
<td>CCTL Command Definition</td>
<td>232</td>
</tr>
<tr>
<td>59</td>
<td>CCTL Commands Which Access <em>mcctldata</em></td>
<td>234</td>
</tr>
<tr>
<td>60</td>
<td>I-Cache CCTL Index Read/Write TAG Bit Fields</td>
<td>235</td>
</tr>
<tr>
<td>61</td>
<td>D-Cache CCTL Index Read/Write TAG Bit Fields</td>
<td>235</td>
</tr>
<tr>
<td>62</td>
<td>User CCTL Command Definition</td>
<td>238</td>
</tr>
<tr>
<td>63</td>
<td>AX45MP-1C NAPOT Range Encoding in PMP Address and Configuration Registers</td>
<td>249</td>
</tr>
<tr>
<td>64</td>
<td>AX45MP-1C NAPOT Range Encoding in PMA Address and Configuration Registers</td>
<td>254</td>
</tr>
<tr>
<td>65</td>
<td>Dual-Issue Capability</td>
<td>262</td>
</tr>
<tr>
<td>66</td>
<td>Load Instruction Throughput and Latency</td>
<td>264</td>
</tr>
<tr>
<td>67</td>
<td>Misaligned Load Throughput and Latency for CPU Revision 5.0.0</td>
<td>265</td>
</tr>
<tr>
<td>68</td>
<td>Divide and Remainder Instruction Throughput and Latency</td>
<td>266</td>
</tr>
<tr>
<td>69</td>
<td>Scalar Floating-Point Instruction Throughput and Latency for CPU Revision 10.0.0 and Later Revisions</td>
<td>269</td>
</tr>
<tr>
<td>70</td>
<td>DSP Instruction Throughput and Latency</td>
<td>270</td>
</tr>
<tr>
<td>71</td>
<td>PLIC Register Summary</td>
<td>275</td>
</tr>
<tr>
<td>72</td>
<td>Meaning of Trigger Type</td>
<td>278</td>
</tr>
<tr>
<td>73</td>
<td>Memory Mapped Machine Timer Registers</td>
<td>282</td>
</tr>
<tr>
<td>74</td>
<td>System Memory Map of DebugModule</td>
<td>286</td>
</tr>
<tr>
<td>75</td>
<td>DMI Memory Map of DebugModule</td>
<td>286</td>
</tr>
</tbody>
</table>
1 Overview

This document describes the programming interface for the AndeStar AX45MP-1C processor.

The organization of this document is as follows: the AX45MP-1C processor is described first, followed by descriptions regarding the associated RISC-V specific platform IP components in Section 18, Section 19 and Section 20.

Note
• The RISC-V spec refers to the hardware thread as “hart”. This is equivalent to “core” herein and they are interchangeably used throughout this document.
• AX45MP-1C is a multicore product, but this data sheet is for the single-core configuration of its full configuration.

1.1 AX45MP-1C Processor Features

The main features of the AX45MP-1C processor are:

Cluster
• Support of 1/2/4 Cores
• Support of MESI cache coherence protocol with Coherence Manager
• Level-2 (L2) cache
  – Shared between cores
  – 16-way, pseudo random replacement
  – Cache size: 0KiB/128KiB/256KiB/512KiB/1MiB/2MiB
  – Non-inclusive non-exclusive policy
  – Cache line size: 64 bytes
  – Multi-Cycle RAM support
  – ECC error protection
  – Hardware prefetch
• Core Interface
  – low-latency/synchronous/asynchronous
• Bus Interface
  – AXI4 Protocol
  – Memory Interface and Memory Mapped I/O (MMIO) Interface
  – Optional I/O Coherence Port (IOCP)
    † Synchronous L2:Bus N:1 clock ratio
  – Configurable data width: 128-bit or 256-bit
  – Configurable address width: 32–47 bits
CPU Core

- 8-stage in-order dual-issue execution pipeline
- Hardware multiplier
  - radix-2/radix-4/radix-16/radix-256/fast
- Hardware divider
- Prediction
  - Dynamic branch prediction
    * 256-entry branch target buffer (BTB)
    * 768-entry branch history table
    * 8-bit global branch history
    * 4-entry return address stack (RAS)
- Machine mode, Supervisor mode and User mode
- Optional performance monitors
- Misaligned memory accesses
- RISC-V physical memory protection (PMP)
- Programmable physical memory attributes (PPMA)

AndeStar V5 ISA

- RISC-V RV64I base integer instruction set
- RISC-V “C” standard extension for compressed instructions
- RISC-V “M” standard extension for integer multiplication and division
- Optional RISC-V RISC-V “P” extension for DSP/SIMD instructions
- RISC-V “A” standard extension for atomic instructions
- Optional RISC-V “N” standard extension for user-level interrupt and exception handling
- Optional RISC-V “F” and “D” standard extensions for single/double-precision floating-point
  - FP16 half-precision floating-point extension
  - Andes BFLOAT16 Extension
- Andes Performance extension
- Andes CoDense extension

Memory Management Unit

- sv39/sv48
- 4/8-entry fully associative ITLB/DTLB
- 32/64/128/256/512-entry 4-way set-associative shared TLB

Memory Subsystem

- Support for non-blocking memory operations
- I & D-Caches
  - I-Cache is virtually indexed and physically tagged
- D-Cache is physically indexed and physically tagged
- Cache size: 8KiB/16KiB/32KiB/64KiB
- Cache line size: 64 bytes
- Set associativity: Direct-mapped/2-way/4-way
- Custom cache control operation through CSR read/write
- D-Cache prefetch support
- D-Cache write-Around support

• I & D local memories
  - Size: 4KiB to 16MiB
  - Interface: RAM

• Memory subsystem soft-error protection
  - Protection scheme: parity-checking or error-checking-and-correction (ECC)
  - Automatic hardware error correction
  - Protected memories:
    * I-Cache tag RAM and data RAM
    * D-Cache tag RAM and data RAM
    * I & D local memories
    * Level-2 cache tag RAM and data RAM

**Power Management**

- Wait-for-interrupt (WFI) mode
- Power Domains

**Debug**

- RISC-V External Debug Support
- Configurable number of breakpoints: 2/4/8
- Optional JTAG or Serial debug port for private debug transport module
  - JTAG: IEEE Std 1149.1 style 4-wire JTAG interface
  - Serial: 2-wire Andes debug interface
- Optional secure debug

**Trace**

- Optional instruction trace

**AndeStar Extension**

- StackSafe hardware stack protection extension
- PowerBrake simple power/performance scaling extension
- Custom performance counter events

**Platform-Level Interrupt Controller (PLIC)**
• Configurable number of interrupts: 1–1023
• Configurable number of interrupt priorities: 3/7/15/31/63/127/255
• Configurable number of targets: 1–16
• Andes Vectored Interrupt extension

1.2 Block Diagram

1.3 Major Components

The following list describes the major components of the ax45mp_core design (the inner core of the AX45MP-1C multi-core processor):

ALU           Arithmetic Logic Unit
BIU           Bus Interface Unit
BPU           Branch Prediction Unit
CSR           Control and Status Register
DCU           Data Cache Unit
DLM           Data Local Memory Controller
1.4 Pipeline Stages and Activities

AX45MP-1C implements an 8-stage dual-issue pipeline architecture. The following figure shows the pipeline stages of the processor.

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>ID</th>
<th>II</th>
<th>EX</th>
<th>MM</th>
<th>LX</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch One</td>
<td>Fetch Two</td>
<td>Instruction Decode</td>
<td>Instruction Issue</td>
<td>Early Execution</td>
<td>Data Memory Access</td>
<td>Late Execution</td>
<td>Instruction Retire and Result Writeback</td>
</tr>
</tbody>
</table>

The pipeline activities of the corresponding stages are:

**F1—Instruction Fetch Stage 1**
- Fetching an instruction block from ILM/Instruction cache/bus
- Dynamic branch prediction

**F2—Instruction Fetch Stage 2**
- Fetch block data replies from ILM/Instruction cache/bus
• Branch prediction target acquired from BPU
• Redirect fetch address to prediction target

ID—Instruction Decode
• Instruction decoding

II—Instruction Issue
• Instruction issue scheduling

EX—Early Execution
• Early ALU instruction execution

MM—Memory Access
• DLM/D-Cache access

LX—Late Execution
• Late ALU instruction execution

WB—Instruction Retire and Result Write-Back
• Interrupt resolution
• Instruction retire
• Register file write back
2 Instruction Set Overview

2.1 Introduction

AX45MP-1C implements The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213 (TD004). The following instruction sets are implemented:

- RV64I base integer instruction set
- RISC-V “C” standard extension
- RISC-V “M” standard extension
- RISC-V “A” standard extension
- RISC-V “F” and “D” standard extensions for single/double-precision floating-point
  - FP16 half-precision floating-point extension
  - Andes BFLOAT16 Extension
- RISC-V “P” extension for DSP/SIMD instructions
- AndeStar V5 instruction extension

For detailed information, please see The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213 (TD004) and AndeStar V5 Instruction Extension Specification (UM165).

2.2 Integer Registers

Table 1 lists all general-purpose integer registers.

<table>
<thead>
<tr>
<th>Register</th>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>zero</td>
<td>Hard-wired zero</td>
</tr>
<tr>
<td>x1</td>
<td>ra</td>
<td>Return address</td>
</tr>
<tr>
<td>x2</td>
<td>sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>x3</td>
<td>gp</td>
<td>Global pointer</td>
</tr>
<tr>
<td>x4</td>
<td>tp</td>
<td>Thread pointer</td>
</tr>
<tr>
<td>x5</td>
<td>t0</td>
<td>Temporary/alternate link register</td>
</tr>
<tr>
<td>x6–x7</td>
<td>t1–t2</td>
<td>Temporaries</td>
</tr>
<tr>
<td>x8</td>
<td>s0/fp</td>
<td>Saved register/frame pointer</td>
</tr>
</tbody>
</table>

Continued on next page...
### 2.3 Atomic Instructions

The RVA extension includes load-reserved/store-conditional and atomic memory operation (AMO) instructions.

#### 2.3.1 Load-Reserved/Store-Conditional Instruction

The processor tracks at most one physical address location for LR-SC instructions at a time. The reservation made by the LR instruction is canceled after any memory operation or exception happens. The address of SC instructions must match the reserved address for SC to succeed.

#### 2.3.2 Atomic Memory Operation Instruction

An atomic memory operation is expanded to LR-modify-SC sequences in the processor. The memory content is first loaded with the LR instruction, then the required operation is performed on the retrieved data, and the final result is written back to the memory by the SC instruction. If the SC instruction fails, the sequence will be retried until it succeeds.

### 2.4 Misaligned Memory Access

AX45MP-1C implements the misaligned memory access to support accessing misaligned addresses without triggering any Address Misaligned exceptions.

By controlling the mmisc_ctl CSR, the scheme can be enabled or disabled. Please see Section 16.12.7 for details.
2.4.1 Exceptions

When the misaligned memory access scheme is enabled, Access Fault exceptions will still be triggered under the following cases:

- Accesses to device regions
- Accesses across ILM or DLM boundary
- Accesses with inconsistency PMA attributes
- Atomic accesses

If the misaligned memory access scheme is disabled, misaligned accesses will trigger Access Fault exceptions or Address Misaligned exceptions. Access fault exceptions are triggered when the following cases occur:

- Atomic accesses
- Address is located in a device region

Other misaligned accesses trigger Address Misaligned exceptions.

2.5 Floating-Point ISA Extension

AX45MP-1C supports the “F” and “D” Standard Extensions for accelerating the performance of floating-point heavy applications. The supported configuration is indicated in the misa (Machine ISA) configuration register.

AX45MP-1C supports the following FPU features:

- Fully pipelined MAC instructions
- Hardware subnormal handling
- All rounding modes

2.5.1 Support for Half-Precision and BFLOAT16 Formats

AX45MP-1C also supports instructions with half-precision (FP16) data type as well as Andes extension instructions for conversion between BFLOAT16 and single-precision formats.

The support for half-precision instructions are implemented by accepting the standard RISC-V floating-point instruction formats with the width field set to “H”.

The support for conversion instructions to/from BFLOAT16 are defined in the AndeStar V5 Instruction Extension Specification (UM165).
2.6 DSP ISA Extension

The processor implements the RISC-V “P” extension (draft) for DSP/SIMD ISA. The supported configuration is indicated in the `mmsc_cfg` register. With the addition of the RISC-V “P” extension (draft), the processor can run various DSP applications with lower power and higher performance.

The supported DSP features include:

- SIMD Data Processing Instructions
- Partial-SIMD Data Processing Instructions
- 64-bit Profile Instructions
- Non-SIMD Instructions
- RV64 Only Instructions
- Overflow Status Manipulation Instructions

Please see the *AndeStar V5 DSP ISA Extension Specification (UM199)* for instruction details.
3 Branch Prediction Unit

The processor implements Branch Prediction Unit (BPU) for branch prediction in instruction fetch. BPU contains a two-way 128-entry branch target buffer (BTB), a 4-entry return address stack (RAS), and a branch history table (BHT).

BTB is implemented to hold target addresses for unconditional jumps and conditional branches. RAS is used to keep return addresses for function calls. BHT performs the taken/not taken prediction for the conditional branches.

Branch predictions can be disabled by setting `mmisc_ctl.BRPE` to 0x0, which will make IFU fetch instructions sequentially without predictions.
4 Memory Management Unit

4.1 Introduction

Memory Management Unit (MMU) is responsible for virtual address to physical address translation. The unit interfaces with the Instruction Fetch Control Unit (IFU) and the Load/Store Unit (LSU). An iTLB is implemented for IFU to speed up instruction address translation, while a dTLB is implemented for LSU to speed up data address translation. If the address translation information is not available in the iTLB or dTLB level, a TLB lookup request will be sent to Shared TLB (STLB), which is a bigger TLB defined in MMU. If a TLB miss happens in STLB, the hardware page table walker (PTW) will automatically traverse through page tables for the translation information.

MMU needs to be enabled and be initialized before it can be used. The default state of MMU is disabled which means there is no virtual to physical address translation. MMU is initialized through the satp CSR. Please see satp for details. Furthermore, SFENCE.VMA will be needed after satp is updated.

4.2 Address Translation

The virtual address to physical address translation is page based. For each virtual page, there is a page table entry (PTE) describing the physical page mapping information. Page table entries are inserted into MMU (iTTLB/dTLB/STLB) by the hardware page table walker (PTW) automatically.
Figure 2: Virtual Address to Physical Address Translation
4.3 Translation Lookaside Buffer

The following sections describe the TLB operations.

4.3.1 Instruction uTLB (iTLB)

Instruction uTLB (iTLB) is a 4/8-entry fully-associative cache that stores address translations (i.e., the Page Table Entries, PTEs) for instruction fetches. The iTLB gets the translation information from STLB under iTLB misses. SFENCE.VMA operations or write to satp will clear all non-global entries in iTLB.

4.3.2 Data uTLB (dTLB)

Data uTLB (dTLB) is a 4/8-entry fully-associative cache that stores address translations (i.e., the Page Table Entries, PTEs) for data accesses. dTLB gets the translation information from STLB under dTLB misses. SFENCE.VMA operations write to satp will clear all non-global entries in dTLB.

4.3.3 Shared TLB (STLB)

The Shared TLB (STLB) contains a 32/64/128/256/512-entry 4-way set-associative structure for 4K pages and a 4-entry fully-associative structure for superpages that store address translations (i.e., the Page Table Entries, PTEs) for both instruction fetches and data accesses.

4.3.4 Replacement Policy

TLBs (iTLB/dTLB/STLB) all implement pseudo-LRU replacement policy.

4.4 Page Table Walker (PTW)

4.4.1 Introduction

The page table walker is responsible for automatically filling STLB with PTE entries located in the system memory when there is a STLB lookup miss. Each TLB miss will require two to four memory references. The implementation caches non-leaf PTEs to speed up page table hierarchy traversal. Two sets of non-leaf PTE caches are implemented—one for misses originating from instruction fetches and the other one for misses originating from data loads/stores.
4.4.2 Page Table Address Formation

To find the correct PTE from the page table in memory, the PTW needs to perform up to three or four memory read operations to get PTEs. The physical addresses for these read operations are formed by the PTW as follows:

1. Let \( a = \text{satp}.ppn \times \text{PAGESIZE} \) and let \( i = \text{LEVELS} - 1 \). (PAGESIZE=\( 2^{12} \) and for Sv39, LEVELS=3; for Sv48, LEVELS=4.)

2. The physical address for PTE of VPN\([i]\) is \( a + \text{va.vpn}[i] \times 8 \). Perform a memory read to retrieve the PTE at this location. PMP checks are also applicable to memory reads accessing the PTEs. If a violation occurs, an access exception will be raised based on the access type of the instructions triggering this page table walk. Let \( pte \) be the value of the memory read if no access violation occurs.

3. If \( pte.v = 0 \), or if \((w, r)\) of \( pte \) is \((1, 0)\), stop and raise a page-fault exception based on the access type of the instructions triggering this page table walk.

4. Otherwise, the PTE is valid.
   - If \((x, w, r)\) of \( pte \) is \((0, 0, 0)\), this PTE is a pointer to the next level of the page table. Let \( i = i - 1 \), \( a = pte.ppn \times \text{PAGESIZE} \) and go to step 2 unless \( i < 0 \), in which case a page-fault exception should be raised based on the access type of the instructions triggering this page table walk.
   - If \((x, w, r)\) of \( pte \) is not \((0, 0, 0)\), then the leaf PTE for the page is found.

4.5 Attributes for Address Spaces

4.5.1 Attributes for Virtual Memory Pages

Each memory page is associated with attributes controlling page accesses. These attributes are stored in the page table entries along with their physical address mappings. The following table describes the format for the page table entries.

<table>
<thead>
<tr>
<th>X</th>
<th>W</th>
<th>R</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Pointer to next level of page table</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Read-only page</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Read-write page</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Execute-only page</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Read-execute page</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>X</th>
<th>W</th>
<th>R</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Read-write-execute page</td>
</tr>
</tbody>
</table>

### Table 2: Translated Address Space Attribute

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td>53:10</td>
<td>Physical Page Number of the physical memory page</td>
</tr>
<tr>
<td>RSW</td>
<td>9:8</td>
<td>The RSW field is reserved for use by supervisor software.</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>The D bit indicates the virtual page has been written since the last time the D bit was cleared. When a virtual page is written and the D bit is clear, a page-fault exception is raised.</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>The A bit indicates the virtual page has been read, written, or fetched from since the last time the A bit was cleared. When a virtual page is accessed and the A bit is clear, a page-fault exception is raised.</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>The G bit designates a global mapping. Global mappings are those that exist in all address spaces. For non-leaf PTEs, the global setting implies that all mappings in the subsequent levels of the page table are global. Note that failing to mark a global mapping as global merely reduces performance, whereas marking a non-global mapping as global is an error.</td>
</tr>
<tr>
<td>U</td>
<td>4</td>
<td>The U bit indicates whether the page is accessible to user mode. U-mode software may only access the page when U=1. If the SUM bit in the status register is set, supervisor mode software may also access pages with U=1. However, supervisor code normally operates with the SUM bit clear, in which case, supervisor code will fault on accesses to user-mode pages.</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td>The X bit indicates whether the page is executable. Attempting to fetch an instruction from a page that does not have execute permissions raises a fetch page-fault exception.</td>
</tr>
<tr>
<td>W</td>
<td>2</td>
<td>The W bit indicates whether the page is writable. Attempting to execute a store, store-conditional (regardless of success), or AMO instruction whose effective address lies within a page without write permissions raises a store page-fault exception.</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>The R bit indicates whether the page is readable. Attempting to execute a load or load-reserved instruction whose effective address lies within a page without read permissions raises a load page-fault exception.</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>The V bit indicates whether the PTE is valid.</td>
</tr>
</tbody>
</table>
5 Local Memory

5.1 Introduction

Local memories store data or instructions that might either be accessed frequently or require deterministic access latency, such as interrupt service routines, system calls, video data, real-time systems, etc. Local memories are memories and accesses to them are treated the same as to the cacheable memory space. It is not suitable to map device registers in the local memories.

AX45MP-1C supports both instruction local memory (ILM) and data local memory (DLM). They are dedicated address spaces that are independent of the memory subsystem. Accesses to them bypass the cache and memory subsystems to achieve minimal latency. The details of local memory usages are described in the subsequent sections.

5.2 Local Memory Spaces

The AX45MP-1C processor supports three address spaces: the instruction local memory, the data local memory and the system bus (AXI) address spaces. The ILM address space is defined by ILM Size and ILM Base configuration options, and the DLM address space is defined by DLM Size and DLM Base configuration options. The base address of the Andes local memory should be aligned to its size (a power-of-2 size). Any addresses outside the local memory address spaces belong to the system bus address space.

Instruction fetches go to the instruction local memory or the system bus while load/store data accesses access all three regions of spaces. The address spaces for ILM and DLM should not overlap with each other to achieve maximum compatibility across Andes processor products. The exact address space access priorities for the AX45MP-1C processor are defined in Table 3 for instruction fetches and Table 4 for load/store data accesses.

It is not recommended to set the instruction local memory and the data local memory to have the same base address. Otherwise, UNPREDICTABLE behavior might happen.

<table>
<thead>
<tr>
<th>Address Hit the ILM Space</th>
<th>Address Hit the DLM Space</th>
<th>Actual Space Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>AXI address space</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>AXI address space</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>ILM</td>
</tr>
</tbody>
</table>

Table 3: Priorities for Instruction Fetches

Continued on next page...
AndesCore AX45MP-1C Processor Reference Manual

Table 3: (continued)

<table>
<thead>
<tr>
<th>Address Hit the ILM Space</th>
<th>Address Hit the DLM Space</th>
<th>Actual Space Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(not recommended; the ILM and DLM spaces should not overlap)</td>
</tr>
</tbody>
</table>

Table 4: Priorities for Data Accesses

<table>
<thead>
<tr>
<th>Address Hit the ILM Space</th>
<th>Address Hit the DLM Space</th>
<th>Actual Space Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>AXI address space</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>DLM</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>ILM</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(not recommended; the ILM and DLM spaces should not overlap)</td>
</tr>
</tbody>
</table>

5.3 Local Memory Address Range

The local memory address ranges are listed in Table 5. LM_BASE represents the base address field of the ILM and DLM local memory base address system registers (milmb.IBPA and mdlmb.DBPA).

Table 5: Local Memory Address Range (for ILM and DLM)

<table>
<thead>
<tr>
<th>LM Size</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>4MiB</td>
<td>(LM_BASE[63:22]&lt;&lt;22)</td>
<td>(LM_BASE[63:22]&lt;&lt;22) + 0x0003FFFFFF</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>LM Size</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
</table>

### 5.4 Local Memory Usage Constraints

Local memories are optimized for access latency. As a result, the design imposes the following usage restrictions:

- Offset addresses of VA and PA should be the same for the part of address offsets that address offset mappings for the offset part of the address that indexes into the local memory.
- Accesses to the local memory are speculative. Devices with side effects on reads should not be mapped to this region.
6 Local Memory Slave Port

6.1 Introduction

The LM slave port enables external bus masters to access the local memories of each core. When an address exceeds ILM/DLM size, the higher address bits are ignored by the slave port. The LM slave port supports FIXED, INCR, and WRAP of AXI burst type and contains five channel FIFOs for read and write accesses. Therefore, all transfers might temporarily be stored in the FIFOs and a round-robin arbiter is used to schedule the read/write access.

Note that AX45MP-1C does not include logics to guarantee atomicity of atomic instructions accessing the LM address space when external masters access the same location through the LM slave port, nor does it provide the protection feature on LM slave port.

6.2 Support for Soft Error Protection

The LM slave port would return bus errors to AXI masters as well as trigger local interrupts to AX45MP-1C when a 2-bit ECC error is detected.

The behavior of ECC logic for local memories is controlled by milmb.ECCEN/mdlmb.ECCEN. The encoding for errors encountered through accesses from the LM slave port is summarized in the table below.

Correctable ECC errors only trigger local interrupts when ECCEN is equal to 3. Uncorrectable ECC errors would trigger local interrupts when ECCEN is equal to 2 or 3. The triggering of local interrupts is controlled by mie.IMECCI and the interrupt status is reported in mip.IMECCI. See Section 16.3.5 and Section 16.3.11 for details.

Data returned through the LM slave port is the ECC corrected version. For uncorrectable ECC errors, bus errors are reported when ECCEN is equal to 2 or 3.

<table>
<thead>
<tr>
<th>ECCEN</th>
<th>Meaning</th>
<th>Data Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable parity/ECC</td>
<td>Uncorrected data</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Generate local interrupts only on uncorrectable parity/ECC errors</td>
<td>Corrected data or bus errors</td>
</tr>
<tr>
<td>3</td>
<td>Generate local interrupts on any type of parity/ECC errors</td>
<td>Corrected data or bus errors</td>
</tr>
</tbody>
</table>
7 Level-1 Caches

7.1 Introduction

AX45MP-1C contains two Level-1 (L1) caches, the instruction cache and the data cache. Both can be configured to 8KiB, 16KiB, 32KiB, or 64KiB in size. The cache line size is fixed to 64 bytes.

The cache organization information can be collected from the micm_cfg register for the instruction cache and the mdcm_cfg register for the data cache. The configuration choices are listed below, and the format of the configuration registers can be found in Section 16.5.1 and Section 16.5.2.

### Table 6: Configuration Choices for the I-Cache

| I-Cache Size | Ways (micm_cfg.
|              | IWAY) | Cache lines per way (micm_cfg.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>ISET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 KiB</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>16 KiB</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>32 KiB</td>
<td>1</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td>64 KiB</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>256</td>
</tr>
</tbody>
</table>

### Table 7: Configuration Choices for the D-Cache

| D-Cache Size | Ways (mdcm_cfg.
|              | DWAY) | Cache lines per way (mdcm_cfg.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>DSET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 KiB</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>32</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 7: (continued)

<table>
<thead>
<tr>
<th>D-Cache Size</th>
<th>Ways</th>
<th>Cache lines per way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(mdcm_cfg.DWAY)</td>
<td>(mdcm_cfg.DSET)</td>
</tr>
<tr>
<td>16 KiB</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>32 KiB</td>
<td>1</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td>64 KiB</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>256</td>
</tr>
</tbody>
</table>

7.2 Cache Replacement Policy

The replacement policy for direct-mapped caches is irrelevant. 2-way and 4-way caches implement random or tree pseudo-LRU replacement policy.

7.3 I-Cache

I-Cache is virtually indexed and physically tagged (VIPT).

7.3.1 I-Cache Fill Operation

The instruction cache fill operation starts when a cacheable line is not in the I-Cache. After a cache miss, a burst read request for the missed cache line is issued to system bus to reduce access time for the missed cache line data. Up to two outstanding requests can be issued to system bus.

The fill operation may be aborted by system bus errors. A precise instruction access fault is triggered for the instruction fetch causing the cache miss operation if the error is on the critical word. If the error occurs on non-critical words, the fill operation will be canceled and the missed line will not be installed into I-Cache. Instruction fetches before non-critical error words will not be affected since they have received the required data.

In Debug Mode, instruction fetches will not affect I-Cache contents, and all I-Cache misses will not cause cache replacements.
### 7.3.2 I-Cache Prefetch

After cache miss, a series of sequential cache lines will be probed to check whether they are in I-Cache. If not, streaming prefetch will be performed for these sequential cache lines. The instruction prefetch is turned off by default, and can be turned on by setting `mcache_ctl.IPREF_EN` to 0x1.

### 7.3.3 I-Cache TAG SRAM Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>[0+:ICACHE_TAG_WIDTH]</td>
<td>TAG</td>
</tr>
<tr>
<td>LOCK_DUP</td>
<td>[ICACHE_TAG_WIDTH]</td>
<td>Duplicated lock bit for error protection</td>
</tr>
<tr>
<td>LOCK</td>
<td>[ICACHE_TAG_WIDTH+1]</td>
<td>Indicate the line is locked</td>
</tr>
<tr>
<td>VALID</td>
<td>[ICACHE_TAG_WIDTH+2]</td>
<td>Indicate the line is valid</td>
</tr>
</tbody>
</table>

**Note**

\[
\text{ICACHE\_TAG\_WIDTH} = \text{BIU\_ADDR\_WIDTH} - \min(12, 6 + \log_2(\text{micm\_cfg.ISET}))
\]
7.4 D-Cache

D-Cache is physically indexed and physically tagged (PIPT).

7.4.1 Cache Access Latency

Table 9: Access Latency of the D-Cache

<table>
<thead>
<tr>
<th>Condition</th>
<th>Load-to-Use Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-Cache Hit</td>
<td>0</td>
</tr>
<tr>
<td>Modified line in another core</td>
<td>57</td>
</tr>
<tr>
<td>L2-Cache hit</td>
<td>24</td>
</tr>
<tr>
<td>L2-Cache miss</td>
<td>27</td>
</tr>
</tbody>
</table>

Note

- The latency assumes that load word instructions are used to access D-Cache.
- The latency assumes that bus latency is one cycle after the request address is issued.
- The latency assumes that the dependent instruction is executed in late ALU.
- The latency assumes that the miss address is the first transfer of a cache line.
- The 0-cycle latency assumes that the dependent instruction and the load instruction are dual-issued.
- The latency assumes that the CORE_CLK and BUS_CLK clock ratio is 1:1.
- The latency assumes that the latency of memory access is 1 cycle.
- The latency assumes that the bus data width is 128-bit.
- The latency assumes that Tag RAM output cycle of L2-Cache is 1 cycle.
- The latency assumes that Tag RAM setup cycle of L2-Cache is 1 cycle.
- The latency assumes that Data RAM output cycle of L2-Cache is 1 cycle.
- The latency assumes that Data RAM setup cycle of L2-Cache is 1 cycle.

7.4.2 D-Cache Fill Operations

The D-Cache fill operation starts when a cacheable line is not in the D-Cache. A burst read request for the missed cache line is always sent first to the system bus to minimize the miss latency. The read request will be followed by a burst write request if cache eviction is required.
The fill operation may be aborted by system bus errors. A precise load access fault is triggered for the load instruction causing the cache miss operation if the bus error is for the critical word. The bus error on non-critical words does not trigger an exception, but the fill operation will be canceled and the missed line will not be installed into D-Cache. Store is always non-blocking. Bus errors on store cache miss will trigger bus write transaction errors.

System bus errors will no longer be reported as precise exceptions if the processor is in the non-blocking mode (mmisc_ctl.NBLD_EN). Bus Read/Write Transaction Error Local Interrupts (mip.BWEI) will be reported instead. Please see Section 11.1 for details.

In Debug Mode, load/store instructions will minimally affect D-Cache contents. All cache misses will not cause cache replacements, and only dirty bits may be affected by accesses to cache lines that are already in D-Cache.

7.4.3 D-Cache Eviction Operations

A burst write request will be sent to L2 if a dirty line is evicted out of D-Cache. An imprecise bus-write error exception is triggered if the burst write request encounters system bus errors.

7.4.4 D-Cache Write Buffers

D-Cache implements two write buffers, and each buffer is used to buffer a 64-byte line. When D-Cache evicts a line, the evicted line is temporarily buffered in one of the write buffers and then send to L2 from that write buffer.

When the Write-Around feature is configured, store to write-no-allocate memory is also buffered in one of the write buffers. Multiple stores can be merged into a single transaction. The buffered line is flushed in any of the following conditions:

- The line is completely filled by subsequent store operations.
- A store operation writes to another line in write-no-allocate memory.
- A load operation accesses the buffered line.
- A FENCE, FENCE.I, or SFENCE.VMA instruction is under execution.
- A CCTL operation is under execution.
- The processor is in debug mode.
- The processor is WFI mode.
- D-Cache is disabled (mcache_ctl.DC_EN is cleared).

Write-no-allocate memory consists of:

- A D-Cache miss, which occurs for a store operation when D-Cache is in the write-no-allocate mode (See Section 11.2).
• PMA of the memory, which is set as (Memory, Write-back, No-allocate/Read-allocate) (MTYP=8/9).

**Note**
When the Write-Around feature is not configured, store to write-no-allocate memory is not buffered.

### 7.4.5 D-Cache TAG SRAM Fields

Table 10: D-Cache TAG SRAM Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESI</td>
<td>[2:0]</td>
<td>Indicate cache line states.</td>
</tr>
<tr>
<td>LOCK</td>
<td>[3]</td>
<td>Indicate the line is locked</td>
</tr>
<tr>
<td>TAG</td>
<td>[4+:DCACHE_TAG_WIDTH]</td>
<td>TAG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td><strong>Invalid</strong>. The line is not in D-Cache.</td>
</tr>
<tr>
<td>001</td>
<td><strong>Shared</strong>. The line is clean, and is shared by multiple caches.</td>
</tr>
<tr>
<td>011</td>
<td><strong>Exclusive</strong>. The cache line is present only in the current cache, but is clean.</td>
</tr>
<tr>
<td>111</td>
<td><strong>Modified</strong>. The line is present only in the current cache, and is dirty.</td>
</tr>
</tbody>
</table>

**Note**

DCACHE_TAG_WIDTH = BIU_ADDR_WIDTH - 6 - log2(mdcm_cfg.DSET)
7.5 FENCE/FENCE.I Operations

FENCE/FENCE.I instructions may affect caches when caches are enabled. If mcache_ctl.IC_EN is 0, FENCE.I instructions will not perform any operation on the I-Cache. If mcache_ctl.DC_EN is 0, FENCE.I instructions will not perform any operation on the D-Cache. FENCE instructions do not perform any operation on either I-Cache or D-Cache. The behavior of FENCE/FENCE.I is summarized in Table 11.

Table 11: Effects of FENCE/FENCE.I Instructions

<table>
<thead>
<tr>
<th>Cache</th>
<th>FENCE</th>
<th>FENCE.I</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Cache</td>
<td>None</td>
<td>Invalidate all cache lines</td>
</tr>
<tr>
<td>D-Cache</td>
<td>None</td>
<td>Write back all cache lines</td>
</tr>
</tbody>
</table>

7.6 CCTL Operations

CCTL operations provide direct control to manipulate instruction or data caches (cache maintenance operations). They are invoked by writing CCTL commands to the mcctlcommand CSR register. The operations can be grouped into two main types, virtual-address (VA) based or index (IX) based, and they are summarized in Table 12. These two addressing types affect how cache lines are specified for CCTL operations, and how the content of mcctlbeginaddr are interpreted.

Table 12: Addressing Type of CCTL Commands

<table>
<thead>
<tr>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX</td>
<td>Use the content of mcctlbeginaddr register directly as a (Way, Index) or (Way, Index, Double-Word/word) pair/tuple to specify a cache line in the cache without going through any translation mechanism. The format is defined in Table 13 and Table 14.</td>
</tr>
<tr>
<td>VA</td>
<td>Use the content of mcctlbeginaddr register as a virtual address to access the cache. The virtual address has to go through the same address translation mechanism in the processor pipeline as the address of regular load/store instructions for D-Cache or instruction fetches for I-Cache. The specified operation is performed only if the addressed cache line is in the corresponding cache.</td>
</tr>
</tbody>
</table>
Table 13: Index Format for D-Cache Index Type of CCTL Operations

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET</td>
<td>mcctlbeginaddr[A-1:3]</td>
<td>A=\log_2(\text{#Double-Words in a Cache Line}) + 3</td>
</tr>
<tr>
<td>INDEX</td>
<td>mcctlbeginaddr[B-1:A]</td>
<td>B=\log_2(\text{Cache Size} / #Ways)</td>
</tr>
<tr>
<td>WAY</td>
<td>mcctlbeginaddr[C-1:B]</td>
<td>C=\text{Ceiling}(\log_2(\text{Cache Size}))</td>
</tr>
</tbody>
</table>

Table 14: Index Format for I-Cache Index Type of CCTL Operations

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET</td>
<td>mcctlbeginaddr[A-1:2]</td>
<td>A=\log_2(\text{#Words in a Cache Line}) + 2</td>
</tr>
<tr>
<td>INDEX</td>
<td>mcctlbeginaddr[B-1:A]</td>
<td>B=\log_2(\text{Cache Size} / #Ways)</td>
</tr>
<tr>
<td>WAY</td>
<td>mcctlbeginaddr[C-1:B]</td>
<td>C=\text{Ceiling}(\log_2(\text{Cache Size}))</td>
</tr>
</tbody>
</table>

The following diagram shows an example of mcctlbeginaddr for the index type of CCTL operations, assuming that cache is 4-way, 32-KiB with 64-byte cache line for D-Cache.

```
63 15 14 13 12  6  5  3  2  0
0  WAY  INDEX  OFFSET 0
```

The following diagram shows an example of mcctlbeginaddr for the index type of CCTL operations, assuming that cache is 4-way, 32-KiB with 64-byte cache line for I-Cache.

```
63 15 14 13 12  6  5  2  1  0
0  WAY  INDEX  OFFSET 0
```

All available CCTL operations are summarized in Table 58. Their detailed definitions are grouped and described in the following categories.

7.6.1 Invalidating Cache Blocks (L1D_VA_INVAL, L1I_VA_INVAL, L1D_IX_INVAL, L1I_IX_INVAL)

These operations invalidate the specified cache lines. Locked cache lines are unlocked and invalidated.

7.6.2 Writing Back Cache Blocks (L1D_VA_WB, L1D_IX_WB, L1D_WB_ALL)

These operations write the data of the specified cache lines back to the system memory, if the specified cache lines are present in the cache with dirty states. The specified cache lines will still be kept in the cache and locked cache lines remain locked.
7.6.3 Writing Back & Invalidating Cache Blocks (L1D_VA_WBINVAL, L1D_Ix_WBINVAL, L1D_WBINVAL_ALL)

These operations write the data of the specified cache lines back to the system memory, if the specified cache lines are present in the cache with dirty states. Then the specified cache lines will be invalidated as long as they are valid in the cache, regardless of whether their states are dirty or locked.

7.6.4 Filling and Locking Cache Blocks (L1D_VA_LOCK, L1I_VA_LOCK)

These operations lock the specified cache lines in the cache. The specified cache lines are first brought into the cache if they are not already present in the cache, then the cache lines are locked by setting their lock states. It is not an error to lock an already locked line—the same line is just locked again.

The lock state only affects the cache replacement policy. On cache miss, an unlocked lines will be replaced first. When all ways are locked, the missed line is not allocated in the cache.

The status of lock operations are written to the mcctl_data register:
• A value of 1 indicates that the lock operation finished successfully;
• A value of 0 indicates that the lock operation aborted/failed.
  – locking an address in device or non-cacheable memory
  – locking an address in local memory (ILM/DLM)
  – locking an line when all ways are locked, and the line is not present in the cache

7.6.5 Unlocking Cache Blocks (L1D_VA_UNLOCK, L1I_VA_UNLOCK)

These operations clear the lock state of the specified cache lines if the specified cache lines are present in the cache.

7.6.6 Reading Tag Data from Caches (L1D_Ix_RTAG, L1I_Ix_RTAG)

These operations read the contents of the tag part of the target cache line into the mcctl_data register. The format of the tag data in mcctl_data is defined in Section 16.12.11. The target cache line is specified in the mcctl_begin_addr register by its way and index information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in mcache_ctl to read the corresponding ECC/Parity codes in the tag RAM to mecc_code.
7.6.7 Reading Data from D-Cache (L1D_IX_RDATA)

This operation reads an 8-byte data from a cache line into the mcctldata register. The target cache line is specified in the mcctlbeginaddr register by its way, index, and double word information. Additionally, this operation observes DC_RWECC settings in mcache_ctl to read the corresponding ECC codes in the data RAM to mecc_code.

7.6.8 Reading Data from I-Caches (L1I_IX_RDATA)

This operation reads a 4-byte data from a cache line into the mcctldata register. The target cache line is specified in the mcctlbeginaddr register by its way, index, and word information. Additionally, this operation observes IC_RWECC settings in mcache_ctl to read the corresponding Parity codes in the data RAM to mecc_code.

7.6.9 Writing Tag Data to Caches (L1D_IX_WTAG, L1I_IX_WTAG)

These operations write the contents of the mcctldata register to the tag part of the target cache line. The format of the tag data in mcctldata is defined in Section 16.12.11. The target cache line is specified in the mcctlbeginaddr register by its way and index information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in mcache_ctl to write the mecc_code ECC/Parity code to the corresponding tag RAM.

---

**Note**

Writing unexpected Tag data to a cache might lead to UNPREDICTABLE behavior when the cache is enable. It is recommended to clear TAG SRAM before enabling the cache.

---

7.6.10 Writing Data to D-Cache (L1D_IX_WDATA)

This operation writes an 8-byte data in the mcctldata register into the target cache line. The target cache line is specified in the mcctlbeginaddr register by its way, index, and double word information. Additionally, this operation observes DC_RWECC settings in mcache_ctl to write the mecc_code ECC code to the corresponding data RAM.
7.6.11 Writing Data to I-Caches (L1I_IX_WDATA)

This operation writes a 4-byte data in the mcctlndata register into the target cache line. The target cache line is specified in the mcctlbeginaddr register by its way, index, and word information. Additionally, this operation observes IC_RWECC settings in mcache_ctl to write the mecc_code Parity code to the corresponding data RAM.

7.6.12 Invalidating All Cache Blocks (L1D_INVAL_ALL)

This operation invalidates all valid lines of D-Cache. Locked cache lines are unlocked and invalidated.

7.6.13 Writing Back All Cache Blocks (L1D_WB_ALL)

This operation writes the data of all dirty cache lines of D-Cache back to the system memory. All cache lines will still remain in the D-Cache and locked lines remain locked.

7.6.14 Writing Back & Invalidating All Cache Blocks (L1D_WBINVAL_ALL)

This operation writes the data of all dirty cache lines of D-Cache back to the system memory and all valid cache lines will be invalidated, including locked and/or clean cache lines.

7.7 Supervisor/User CCTL Operations

CCTL operations are available to Supervisor/User-mode software under the control of the mcacheCtl.CCTL_SUEN control bit. These operations are triggered in both modes by accessing ucctlbeginaddr, ucctlcommand and scctlldata registers, while mcache_ctl.CCTL_SUEN controls access permission to these registers. When CCTL_SUEN is 0, accessing them in Supervisor and User mode would cause illegal instruction exceptions. It should be set to 1 to enable Supervisor (and User) CCTL operations.

All CCTL operations can be made available to Supervisor-mode software while only four CCTL operations listed in the table below are available to User-mode software.

<table>
<thead>
<tr>
<th>Value</th>
<th>Command</th>
<th>Type</th>
<th>Exception Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b00_000</td>
<td>L1D_VA_INVAL</td>
<td>VA</td>
</tr>
<tr>
<td>1</td>
<td>0b00_001</td>
<td>L1D_VA_WB</td>
<td>VA</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Value</th>
<th>Command</th>
<th>Type</th>
<th>Exception Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0b00_010</td>
<td>VA</td>
<td>Store related Fault</td>
</tr>
<tr>
<td>8</td>
<td>0b01_000</td>
<td>VA</td>
<td>Store related Fault</td>
</tr>
</tbody>
</table>
8 Level-2 Cache

8.1 Introduction

A level-2 cache (L2C) is used to improve the system performance by providing larger amount of cache line entries and reasonable access delays. The L2C is shared between cores, and non-inclusive non-exclusive policy is used.

8.1.1 Multi-Cycle RAM Support

Programmable RAM cycles enable L2C to utilize slow RAMs which have worse access timings. For tag and data RAMs respectively, two programmable settings are provided: setup cycle control and data output cycle control.

Setup Cycle Control The setup cycle control indicates the number of clock cycles that the RAM control/data signals should remain valid prior to the RAM clock edge at which the clock enable signal is high. Two types of clock cycles are supported, and their values as well as meanings are listed in Table 16.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>One-cycle (single-cycle). The control and data input signals just need to be valid at the same cycle of the clock enable signal. This is the normal single-cycle RAM usage.</td>
</tr>
<tr>
<td>0x1</td>
<td>Two-cycle. The control and data input signals need to be valid one cycle earlier than the cycle of the clock enable signal.</td>
</tr>
</tbody>
</table>

Data Output Cycle Control Data output cycle control indicates the number of clock cycles taken by the read data to become valid after the RAM clock edge at which the clock enable signal is high. Three types of output clock cycles are supported, and their values as well as meanings are listed in Table 17.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>One-cycle (single-cycle). The data output can be sampled at the subsequent active clock edge.</td>
</tr>
<tr>
<td>0x1</td>
<td>Two-cycle. The data output can be sampled at the second active clock edge.</td>
</tr>
</tbody>
</table>

Continued on next page...
### Table 17: (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2</td>
<td>Three-cycle. The data output can be sampled at the third active clock edge.</td>
</tr>
<tr>
<td>Others</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

#### 8.1.2 L2-Cache Prefetch

L2C implements a hardware prefetch engine, and the key features of the L2-Cache prefetch include:

- Non-unit stride detection for load-store misses within a 4K page
- Consecutive cache line fetches on L2 instruction fetch misses or L2-Cache prefetch hits
- Programmable prefetch depth of instruction and data
  - 0, 1, 2, 3 prefetch requests for instruction
  - 0, 2, 4, 8 prefetch requests for data
- Programmable threshold of fill/evict buffer count to adjust the number of prefetch requests

#### 8.1.3 Asynchronous Error

L2C may encounter errors that cannot be reported to the upstream processors directly through responses of bus transactions, for example, errors hit by its prefetcher. These errors will be reported asynchronously through the interrupt controller (PLIC) as external interrupts. These interrupts could be cleared by writing one to the corresponding fields of the L2C Asynchronous Error Register. Please see Section 8.4.9 for details.

#### 8.1.4 Cache Control Operation

L2C provides a set of cache control operations, which are divided into four types as below:

- IX type (index)
- PA type (physical address)
- TGT type (target)
- Flush type

Table 18 lists the supported L2C CCTL operations. These CCTL operations are performed/controlled by writing the related CCTL control registers. The result of CCTL operations is reported in the CCTL Status Register.
Table 18: Supported L2C CCTL Operations

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>Mnemonics</th>
<th>Type</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00_000</td>
<td>L2_IX_INVAL</td>
<td>IX</td>
<td>Invalidate an L2-Cache entry</td>
</tr>
<tr>
<td>0b00_001</td>
<td>L2_IX_WB</td>
<td>IX</td>
<td>Write-back an L2-Cache entry</td>
</tr>
<tr>
<td>0b00_010</td>
<td>L2_IX_WBINVAL</td>
<td>IX</td>
<td>Write-back and invalidate an L2-Cache entry</td>
</tr>
<tr>
<td>0b01_000</td>
<td>L2_PA_INVAL</td>
<td>PA</td>
<td>Invalidate an L2-Cache entry</td>
</tr>
<tr>
<td>0b01_001</td>
<td>L2_PA_WB</td>
<td>PA</td>
<td>Write-back an L2-Cache entry</td>
</tr>
<tr>
<td>0b01_010</td>
<td>L2_PA_WBINVAL</td>
<td>PA</td>
<td>Write-back and invalidate an L2-Cache entry</td>
</tr>
<tr>
<td>0b10_000</td>
<td>L2_TGT_WRITE</td>
<td>TGT</td>
<td>Write an L2-Cache entry</td>
</tr>
<tr>
<td>0b10_001</td>
<td>L2_TGT_READ</td>
<td>TGT</td>
<td>Read an L2-Cache entry</td>
</tr>
<tr>
<td>0b10_010</td>
<td>L2_WBINVAL_ALL</td>
<td>Flush</td>
<td>Write-back and invalidate all L2-Cache entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Cache flush)</td>
</tr>
</tbody>
</table>

\[L2\_TGT\_WRITE\] and [L2\_TGT\_READ] are used to directly write and read the raw data of a tag or data RAM entry. If L2C is configured to support ECC, the ECC code of the RAM entry will also be written or read through the CCTL TGT Write ECC Code Register or CCTL TGT Read ECC Code Register. For raw data, this means the ECC checking will not be performed during the CCTL operation when ECC is enabled. However, if [L2\_TGT\_WRITE] is used to write a corrupted ECC code for a valid tag or data RAM entry, the ECC error will be detected when this entry is later used by normal cache operations while ECC is enabled.

### 8.2 L2-Cache Organization

The L2-Cache implements a 16-way set-associative design with the pseudo-random cache-replacement policy. The cache consists of data and tag RAMs. The data RAM stores the cache line data, while the tag RAM stores the corresponding status and address information.

#### 8.2.1 Two Bank L2-Cache Structure

Both tag and data RAMs of the L2-Cache are partitioned into two banks to support parallel operations.

### 8.3 L2C Programming Guide

The enablement of the L2-Cache is controlled by the CEN field of the L2C control register. To properly enable the L2-Cache to cache both instructions and data, apply the following sequence:
1. Configure related L2C functions by writing the L2C control register with desired values for the 
   PFTHRES, IPFDPT, DPFDPT, ECCEN, TRAMOCTL, TRAMICTL, DRAMOCTL and DRAMICTL fields. 
   The CEN field should be set to 0. This step can be skipped if the resulting write value is 0.
2. Write the L2C control register again using the same value of step 1 with the CEN field being 1.

To disable the L2-Cache to stop caching both instructions and data, apply the following sequence:
1. Write 0 to the CEN field of the L2C control register.
2. Write-back and invalidate the L2-Cache by using the CCTL flush command (L2_WBINVAL_ALL) 
   as described in Section 8.4.11.

8.4 L2-Cache Controller Registers

L2C controller registers are memory mapped to a 1MiB space for access through normal RISC-V load-
/store instructions. The base address of this 1MiB space is decided by the L2C Register Base option. 
The cacheability of bus transactions to this region is ignored and only SINGLE transactions are sup-
ported. The result of using other burst types of transactions to access this region is UNPREDICTABLE.

8.4.1 Register Type

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>Implementation dependent/determined</td>
</tr>
<tr>
<td>RO</td>
<td>Read-Only register/field. Any software write to RO registers/fields will be silently ignored by hardware.</td>
</tr>
<tr>
<td>RW</td>
<td>Read/Write register/field</td>
</tr>
<tr>
<td>W1C</td>
<td>Write-One-Clear register/field. When written 1, the corresponding bit of the register/field is cleared to 0.</td>
</tr>
<tr>
<td>WC</td>
<td>Write-Clear register/field. When written any value, the register/field is cleared to 0.</td>
</tr>
</tbody>
</table>

8.4.2 Summary of Registers

The summary of registers is shown in Table 19.

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 – 0x0007</td>
<td>Configuration Register</td>
<td>Section 8.4.3</td>
</tr>
<tr>
<td>0x0008 – 0x000f</td>
<td>Control Register</td>
<td>Section 8.4.4</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0010 – 0x0017</td>
<td>HPM Control Register 0</td>
<td>Section 8.4.5</td>
</tr>
<tr>
<td>0x0018 – 0x001f</td>
<td>HPM Control Register 1</td>
<td>Section 8.4.6</td>
</tr>
<tr>
<td>0x0020 – 0x0027</td>
<td>HPM Control Register 2</td>
<td>Section 8.4.7</td>
</tr>
<tr>
<td>0x0028 – 0x002f</td>
<td>HPM Control Register 3</td>
<td>Section 8.4.8</td>
</tr>
<tr>
<td>0x0030 – 0x0037</td>
<td>Asynchronous Error Register</td>
<td>Section 8.4.9</td>
</tr>
<tr>
<td>0x0038 – 0x003f</td>
<td>Error Register</td>
<td>Section 8.4.10</td>
</tr>
<tr>
<td>0x0040 – 0x0047</td>
<td>Core 0 CCTL Command Register</td>
<td>Section 8.4.11</td>
</tr>
<tr>
<td>0x0048 – 0x004f</td>
<td>Core 0 CCTL Access Line Register</td>
<td>Section 8.4.12</td>
</tr>
<tr>
<td>0x0050 – 0x0057</td>
<td>Core 1 CCTL Command Register</td>
<td>Section 8.4.11</td>
</tr>
<tr>
<td>0x0058 – 0x005f</td>
<td>Core 1 CCTL Access Line Register</td>
<td>Section 8.4.12</td>
</tr>
<tr>
<td>0x0060 – 0x0067</td>
<td>Core 2 CCTL Command Register</td>
<td>Section 8.4.11</td>
</tr>
<tr>
<td>0x0068 – 0x006f</td>
<td>Core 2 CCTL Access Line Register</td>
<td>Section 8.4.12</td>
</tr>
<tr>
<td>0x0070 – 0x0077</td>
<td>Core 3 CCTL Command Register</td>
<td>Section 8.4.11</td>
</tr>
<tr>
<td>0x0078 – 0x007f</td>
<td>Core 3 CCTL Access Line Register</td>
<td>Section 8.4.12</td>
</tr>
<tr>
<td>0x0080 – 0x0087</td>
<td>CCTL Status Register</td>
<td>Section 8.4.13</td>
</tr>
<tr>
<td>0x0088 – 0x008f</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x0090 – 0x00c8</td>
<td>CCTL TGT Data Register 0 to 7</td>
<td>Section 8.4.14</td>
</tr>
<tr>
<td>0x00d0 – 0x00d7</td>
<td>CCTL TGT ECC Code Register</td>
<td>Section 8.4.15</td>
</tr>
<tr>
<td>0x0200 – 0x02f8</td>
<td>HPM Counter Register 0 to 31</td>
<td>Section 8.4.16</td>
</tr>
</tbody>
</table>
8.4.3 Configuration Register

Offset: 0x0000

This register indicates the cache size, ECC type, and the version of the L2C implementation.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00</td>
<td>0 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x01</td>
<td>128 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x02</td>
<td>256 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x04</td>
<td>512 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x08</td>
<td>1024 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x10</td>
<td>2048 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Others</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>[19:16]</td>
<td>L2C ECC type</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ECC Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0</td>
<td>No protection</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>One-bit error correction and two-bit error detection</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Others</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>VERSION</td>
<td>[31:24]</td>
<td>L2C version</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>
8.4.4 Control Register

Offset: 0x0008

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEN</td>
<td>[0]</td>
<td>L2-Cache enable</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable the L2-Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enable the L2-Cache</td>
<td></td>
</tr>
<tr>
<td>PFTHRES</td>
<td>[2:1]</td>
<td>Prefetch threshold value. To avoid occupying too many L2C internal line buffers, prefetch can be throttled based on occupancy status of line buffers.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable L2C prefetch requests when the number of used line buffers exceeds 10.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Disable L2C prefetch requests when the number of used line buffers exceeds 8.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Disable L2C prefetch requests when the number of used line buffers exceeds 6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Never disable L2C prefetch requests regardless of the number of used line buffers.</td>
<td></td>
</tr>
</tbody>
</table>

This register controls various functions of L2C.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPFDPT</td>
<td>[4:3]</td>
<td>Instruction prefetch depth. Indicates the depth of L2 instruction prefetch. It is the number of prefetch requests ahead of the demand request stream.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 request (disable L2 instruction prefetch)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 requests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 requests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPFDPT</td>
<td>[6:5]</td>
<td>Data prefetch depth. Indicates the depth of L2 data prefetch. It is the number of prefetch requests ahead of the demand request stream.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 request (disable L2 data prefetch)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 requests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4 requests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8 requests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECCEN</td>
<td>[7]</td>
<td>ECC enable</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disable L2 ECC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable L2 ECC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note**
This field is only valid when L2C ECC is configured.

Continued on next page...
### Field Name: TRAMOCTL [9:8]

Tag RAM output cycle control. See Section 8.1.1 for more details.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 cycle</td>
</tr>
<tr>
<td>1</td>
<td>2 cycles</td>
</tr>
<tr>
<td>2</td>
<td>3 cycles</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Field Name: TRAMICTL [10]

Tag RAM setup cycle control. See Section 8.1.1 for more details.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 cycle</td>
</tr>
<tr>
<td>1</td>
<td>2 cycles</td>
</tr>
</tbody>
</table>

### Field Name: DRAMOCTL [12:11]

Data RAM output cycle control. See Section 8.1.1 for more details.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 cycle</td>
</tr>
<tr>
<td>1</td>
<td>2 cycles</td>
</tr>
<tr>
<td>2</td>
<td>3 cycles</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Field Name: DRAMICTL [13]

Data RAM setup cycle control. See Section 8.1.1 for more details.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 cycle</td>
</tr>
<tr>
<td>1</td>
<td>2 cycles</td>
</tr>
</tbody>
</table>

### Field Name: INITSTATUS [14]

L2C self initialization status

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initialization done</td>
</tr>
<tr>
<td>1</td>
<td>Under initialization</td>
</tr>
</tbody>
</table>
8.4.5 HPM Control Register 0

Offset: 0x0010

This register selects events to monitor for performance counters 0–7. It is only present when the number of configured L2C Performance Counter is greater than 0.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL0</td>
<td>[7:0]</td>
<td>Monitored event selection of performance counter 0</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL1</td>
<td>[15:8]</td>
<td>Monitored event selection of performance counter 1</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL2</td>
<td>[23:16]</td>
<td>Monitored event selection of performance counter 2</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL3</td>
<td>[31:24]</td>
<td>Monitored event selection of performance counter 3</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL5</td>
<td>[47:40]</td>
<td>Monitored event selection of performance counter 5</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL6</td>
<td>[55:48]</td>
<td>Monitored event selection of performance counter 6</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL7</td>
<td>[63:56]</td>
<td>Monitored event selection of performance counter 7</td>
<td>RW</td>
<td>0xff</td>
</tr>
</tbody>
</table>

Table 20 lists the supported events of performance counters.

<table>
<thead>
<tr>
<th>Select</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Total access count</td>
</tr>
<tr>
<td>0x01</td>
<td>L2-Cache access count</td>
</tr>
<tr>
<td>0x02</td>
<td>L2-Cache miss count</td>
</tr>
<tr>
<td>0x03-0x0f</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x10-0x4f</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x50</td>
<td>Number of requests sent to the system bus</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 20: (continued)

<table>
<thead>
<tr>
<th>Select</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x51-0x6f</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x70</td>
<td>L2 way 0 eviction count</td>
</tr>
<tr>
<td>0x71</td>
<td>L2 way 1 eviction count</td>
</tr>
<tr>
<td>0x72</td>
<td>L2 way 2 eviction count</td>
</tr>
<tr>
<td>0x73</td>
<td>L2 way 3 eviction count</td>
</tr>
<tr>
<td>0x74</td>
<td>L2 way 4 eviction count</td>
</tr>
<tr>
<td>0x75</td>
<td>L2 way 5 eviction count</td>
</tr>
<tr>
<td>0x76</td>
<td>L2 way 6 eviction count</td>
</tr>
<tr>
<td>0x77</td>
<td>L2 way 7 eviction count</td>
</tr>
<tr>
<td>0x78</td>
<td>L2 way 8 eviction count</td>
</tr>
<tr>
<td>0x79</td>
<td>L2 way 9 eviction count</td>
</tr>
<tr>
<td>0x7a</td>
<td>L2 way 10 eviction count</td>
</tr>
<tr>
<td>0x7b</td>
<td>L2 way 11 eviction count</td>
</tr>
<tr>
<td>0x7c</td>
<td>L2 way 12 eviction count</td>
</tr>
<tr>
<td>0x7d</td>
<td>L2 way 13 eviction count</td>
</tr>
<tr>
<td>0x7e</td>
<td>L2 way 14 eviction count</td>
</tr>
<tr>
<td>0x7f</td>
<td>L2 way 15 eviction count</td>
</tr>
<tr>
<td>0x80-0xff</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
8.4.6 HPM Control Register 1

Offset: 0x0018

This register selects the events to monitor for performance counters 8–15. It is only present when the number of configured L2C Performance Counter is greater than 8.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL8</td>
<td>[7:0]</td>
<td>Monitored event selection of performance counter 8</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL9</td>
<td>[15:8]</td>
<td>Monitored event selection of performance counter 9</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL10</td>
<td>[23:16]</td>
<td>Monitored event selection of performance counter 10</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL11</td>
<td>[31:24]</td>
<td>Monitored event selection of performance counter 11</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL13</td>
<td>[47:40]</td>
<td>Monitored event selection of performance counter 13</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL14</td>
<td>[55:48]</td>
<td>Monitored event selection of performance counter 14</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL15</td>
<td>[63:56]</td>
<td>Monitored event selection of performance counter 15</td>
<td>RW</td>
<td>0xff</td>
</tr>
</tbody>
</table>
8.4.7 HPM Control Register 2

Offset: 0x0020

This register selects the events to monitor for performance counters 16–23. It is only present when the number of configured L2C Performance Counter is greater than 16.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL16</td>
<td>[7:0]</td>
<td>Monitored event selection of performance counter 16</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL17</td>
<td>[15:8]</td>
<td>Monitored event selection of performance counter 17</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL18</td>
<td>[23:16]</td>
<td>Monitored event selection of performance counter 18</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL19</td>
<td>[31:24]</td>
<td>Monitored event selection of performance counter 19</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL21</td>
<td>[47:40]</td>
<td>Monitored event selection of performance counter 21</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL22</td>
<td>[55:48]</td>
<td>Monitored event selection of performance counter 22</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL23</td>
<td>[63:56]</td>
<td>Monitored event selection of performance counter 23</td>
<td>RW</td>
<td>0xff</td>
</tr>
</tbody>
</table>
8.4.8  HPM Control Register 3

Offset: 0x0028

This register selects the events to monitor for performance counters 24–31. It is only present when the number of configured L2C Performance Counter is greater than 24.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL24</td>
<td>[7:0]</td>
<td>Monitored event selection of performance counter 24</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL25</td>
<td>[15:8]</td>
<td>Monitored event selection of performance counter 25</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL26</td>
<td>[23:16]</td>
<td>Monitored event selection of performance counter 26</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL27</td>
<td>[31:24]</td>
<td>Monitored event selection of performance counter 27</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL29</td>
<td>[47:40]</td>
<td>Monitored event selection of performance counter 29</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL30</td>
<td>[55:48]</td>
<td>Monitored event selection of performance counter 30</td>
<td>RW</td>
<td>0xff</td>
</tr>
<tr>
<td>SEL31</td>
<td>[63:56]</td>
<td>Monitored event selection of performance counter 31</td>
<td>RW</td>
<td>0xff</td>
</tr>
</tbody>
</table>
8.4.9 Asynchronous Error Register

Offset: 0x0030

This register is write-clear and indicates the occurrence of asynchronous errors of L2C.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYNCERR</td>
<td>[0]</td>
<td>Indicates if any asynchronous error has happened</td>
<td>WC</td>
<td>0x0</td>
</tr>
</tbody>
</table>
8.4.10 Error Register

Offset: 0x0038

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>[15:0]</td>
<td>Index address of the first error</td>
<td>WC</td>
<td>0</td>
</tr>
<tr>
<td>WAY</td>
<td>[19:16]</td>
<td>Way of the RAM, where the first error occurred</td>
<td>WC</td>
<td>0</td>
</tr>
</tbody>
</table>

This register holds information of the first error and whether more errors occurred after the first error since the last clearing of this register. L2C-visible errors are divided into two categories, memory errors and bus errors. Memory errors include all errors happening in L1 and L2 RAMs. Bus errors happen in the AXI interface for the next-level memory. The INDEX and WAY fields are only meaningful when the first error is a memory error and occurs in L2 RAMs. If the first error is a memory error in L1 RAMs or is a bus error, the INDEX and WAY fields are “Don’t Care”. Writing any value to this register clears the whole register to zero.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMID</td>
<td>[23:20]</td>
<td>RAM ID, where the first error occurred</td>
<td>WC</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>L2 tag RAM</td>
</tr>
<tr>
<td>0b0001</td>
<td>L2 data RAM</td>
</tr>
<tr>
<td>0b0100</td>
<td>D-Cache tag RAM of CPU core 0</td>
</tr>
<tr>
<td>0b0101</td>
<td>D-Cache tag RAM of CPU core 1</td>
</tr>
<tr>
<td>0b0110</td>
<td>D-Cache tag RAM of CPU core 2</td>
</tr>
<tr>
<td>0b0111</td>
<td>D-Cache tag RAM of CPU core 3</td>
</tr>
<tr>
<td>0b1100</td>
<td>D-Cache data RAM of CPU core 0</td>
</tr>
<tr>
<td>0b1101</td>
<td>D-Cache data RAM of CPU core 1</td>
</tr>
<tr>
<td>0b1110</td>
<td>D-Cache data RAM of CPU core 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>D-Cache data RAM of CPU core 3</td>
</tr>
</tbody>
</table>

| Others   | Reserved                     |

| BUSERR    | [29]  | Set to 1 when the first error is a bus error, and 0 when the first error is a memory error. | WC   | 0     |

| MORERR    | [30]  | More error indicator, set to 0 on the first error and 1 when more errors occurred. | WC   | 0     |

| VALID     | [31]  | Valid bit, set to 1 when the first error (bus or memory) occurs. | WC   | 0     |
8.4.11  CCTL Command Registers 0 – 3

**Offset:** 0x0040, 0x0050, 0x0060, 0x0070

Writing to CCTL Command Register \( n \) (\( n = 0 – 3 \)) will trigger a CCTL operation, with the operation specified by the value written and the line specified by CCTL Access Line Register \( n \). Each register corresponds to one CPU core so each CPU core should only use its respective registers. If CPU core \( n \) does not exist, CCTL Command Register \( n \) is reserved. L2C internally uses the round-robin strategy to decide the run order when two or more CCTL Command Registers are written at the same time.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>[4:0]</td>
<td>CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
8.4.12 CCTL Access Line Registers 0 – 3

Offset: 0x0048, 0x0058, 0x0068, 0x0078

These registers are used to specify the physical address, set, way, and/or RAM ID for CCTL operations. Each register corresponds to one CPU core so each CPU core should only use its respective register. If CPU core \( n (n=0 – 3) \) does not exist, CCTL Access Line Register \( n \) is reserved.

The interpretation of these registers is different based on the CCTL operation type. The register format for IX-type CCTL operations is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>[19:6]</td>
<td>Cache set of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>WAY</td>
<td>[31:28]</td>
<td>Cache way of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

The register format for PA-type CCTL operations is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA</td>
<td>[63:0]</td>
<td>Physical address of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

The register format for TGT-type CCTL operations is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>[19:6]</td>
<td>Cache set of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>RAMID</td>
<td>[27:26]</td>
<td>RAM ID of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>L2 tag RAM</td>
</tr>
<tr>
<td>0b01</td>
<td>L2 data RAM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAY</td>
<td>[31:28]</td>
<td>Cache way of the CCTL operation</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
As for the flush type, these registers are not used.
8.4.13 CCTL Status Register

Offset: 0x0080

This register provides the status of CCTL operations.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS0</td>
<td>[3:0]</td>
<td>CCTL status of Master (Core) 0</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0000</td>
<td>Idle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0001</td>
<td>A CCTL operation is running</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0010</td>
<td>An illegal CCTL operation was performed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATUS1</td>
<td>[7:4]</td>
<td>CCTL status of Master (Core) 1</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>STATUS2</td>
<td>[11:8]</td>
<td>CCTL status of Master (Core) 2</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>STATUS3</td>
<td>[15:12]</td>
<td>CCTL status of Master (Core) 3</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>
8.4.14 CCTL TGT Data Registers 0 – 7

**Offset:** 0x0090 to 0x00c8

These registers provide 512-bit write or read data for the `L2_TGT_WRITE` or `L2_TGT_READ` CCTL operation. They consist of eight 64-bit registers which are read/writable as below:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x90</td>
<td>CCTL TGT Data Register 0 (bits [63:0])</td>
</tr>
<tr>
<td>0x98</td>
<td>CCTL TGT Data Register 1 (bits [127:64])</td>
</tr>
<tr>
<td>0xa0</td>
<td>CCTL TGT Data Register 2 (bits [191:128])</td>
</tr>
<tr>
<td>0xa8</td>
<td>CCTL TGT Data Register 3 (bits [255:192])</td>
</tr>
<tr>
<td>0xb0</td>
<td>CCTL TGT Data Register 4 (bits [319:256])</td>
</tr>
<tr>
<td>0xb8</td>
<td>CCTL TGT Data Register 5 (bits [383:320])</td>
</tr>
<tr>
<td>0xc0</td>
<td>CCTL TGT Data Register 6 (bits [447:384])</td>
</tr>
<tr>
<td>0xc8</td>
<td>CCTL TGT Data Register 7 (bits [511:448])</td>
</tr>
</tbody>
</table>

The interpretation of these registers is different based on the TGT operation. The register format for `L2_TGT_READ/L2_TGT_WRITE` accessing tag RAMs is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>[TAG_DW-1:0]</td>
<td>Tag value of the tag RAM entry</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>VALID</td>
<td>[TAG_DW]</td>
<td>Valid bit of the tag RAM entry</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>DIRTY</td>
<td>[TAG_DW+1]</td>
<td>Dirty bit of the tag RAM entry</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

The data width of tag is shown in Table 21.

<table>
<thead>
<tr>
<th>Size (KiB)</th>
<th>L2C_TAG_DW</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>BIU_ADDR_WIDTH - 13</td>
</tr>
<tr>
<td>256</td>
<td>BIU_ADDR_WIDTH - 14</td>
</tr>
<tr>
<td>512</td>
<td>BIU_ADDR_WIDTH - 15</td>
</tr>
</tbody>
</table>

The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.
Table 21: (continued)

<table>
<thead>
<tr>
<th>Size (KiB)</th>
<th>L2C_TAG_DW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>BIU_ADDR_WIDTH - 16</td>
</tr>
<tr>
<td>2048</td>
<td>BIU_ADDR_WIDTH - 17</td>
</tr>
</tbody>
</table>

The register format for L2_TGT_READ/L2_TGT_WRITE accessing data RAMs is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>[63:0]</td>
<td>Read data from data RAM / Write data to data RAM</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

If an L2_TGT_WRITE operation is applied to the data RAM, the corresponding line of the data RAM is filled with the content of CCTL Data Registers 0 – 7. Otherwise, the corresponding line of the tag RAM is filled.

If an L2_TGT_READ operation is applied to the data RAM, CCTL Data Registers 0 – 7 are filled with the corresponding line of the data RAM. Otherwise, they are filled with the corresponding line of the tag RAM.
8.4.15 CCTL TGT ECC Code Register

Offset: 0x00d0

This register is used to provide ECC codes accompanying the write or read data when the `L2_TGT_WRITE` or `L2_TGT_READ` CCTL operation is performed.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGTECC0</td>
<td>[7:0]</td>
<td>ECC code of CCTL TGT data 0</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC1</td>
<td>[15:8]</td>
<td>ECC code of CCTL TGT data 1</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC2</td>
<td>[23:16]</td>
<td>ECC code of CCTL TGT data 2</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC3</td>
<td>[31:24]</td>
<td>ECC code of CCTL TGT data 3</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC4</td>
<td>[39:32]</td>
<td>ECC code of CCTL TGT data 4</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC5</td>
<td>[47:40]</td>
<td>ECC code of CCTL TGT data 5</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC6</td>
<td>[55:48]</td>
<td>ECC code of CCTL TGT data 6</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>TGTECC7</td>
<td>[63:56]</td>
<td>ECC code of CCTL TGT data 7</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
8.4.16 HPM Counter Registers 0 – 31

Offset: 0x0200 to 0x02F8

These read/writable registers provide the counter values of monitored events. The L2C Performance Counter Number configuration option determines how many of them are present.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0200</td>
<td>Performance Counter 0</td>
</tr>
<tr>
<td>0x0208</td>
<td>Performance Counter 1</td>
</tr>
<tr>
<td>0x0210</td>
<td>Performance Counter 2</td>
</tr>
<tr>
<td>0x0218</td>
<td>Performance Counter 3</td>
</tr>
<tr>
<td>0x0220</td>
<td>Performance Counter 4</td>
</tr>
<tr>
<td>0x0228</td>
<td>Performance Counter 5</td>
</tr>
<tr>
<td>0x0230</td>
<td>Performance Counter 6</td>
</tr>
<tr>
<td>0x0238</td>
<td>Performance Counter 7</td>
</tr>
<tr>
<td>0x0240</td>
<td>Performance Counter 8</td>
</tr>
<tr>
<td>0x0248</td>
<td>Performance Counter 9</td>
</tr>
<tr>
<td>0x0250</td>
<td>Performance Counter 10</td>
</tr>
<tr>
<td>0x0258</td>
<td>Performance Counter 11</td>
</tr>
<tr>
<td>0x0260</td>
<td>Performance Counter 12</td>
</tr>
<tr>
<td>0x0268</td>
<td>Performance Counter 13</td>
</tr>
<tr>
<td>0x0270</td>
<td>Performance Counter 14</td>
</tr>
<tr>
<td>0x0278</td>
<td>Performance Counter 15</td>
</tr>
<tr>
<td>0x0280</td>
<td>Performance Counter 16</td>
</tr>
<tr>
<td>0x0288</td>
<td>Performance Counter 17</td>
</tr>
<tr>
<td>0x0290</td>
<td>Performance Counter 18</td>
</tr>
<tr>
<td>0x0298</td>
<td>Performance Counter 19</td>
</tr>
<tr>
<td>0x02a0</td>
<td>Performance Counter 20</td>
</tr>
<tr>
<td>0x02a8</td>
<td>Performance Counter 21</td>
</tr>
<tr>
<td>0x02b0</td>
<td>Performance Counter 22</td>
</tr>
<tr>
<td>0x02b8</td>
<td>Performance Counter 23</td>
</tr>
<tr>
<td>0x02c0</td>
<td>Performance Counter 24</td>
</tr>
<tr>
<td>0x02c8</td>
<td>Performance Counter 25</td>
</tr>
<tr>
<td>0x02d0</td>
<td>Performance Counter 26</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02d8</td>
<td>Performance Counter 27</td>
</tr>
<tr>
<td>0x02e0</td>
<td>Performance Counter 28</td>
</tr>
<tr>
<td>0x02e8</td>
<td>Performance Counter 29</td>
</tr>
<tr>
<td>0x02f0</td>
<td>Performance Counter 30</td>
</tr>
<tr>
<td>0x02f8</td>
<td>Performance Counter 31</td>
</tr>
</tbody>
</table>
9 L1 Memory Error Protection

AX45MP-1C provides ECC (error correction code) and parity protection schemes to protect RAMs against soft errors. Soft errors are caused by some external particles that temporarily corrupts value stored in the RAMs, and can be recovered or detected with different protection schemes on the RAMs. Supported protection schemes include:

- Parity for I-Cache: Single Error Detection (SED),
- ECC for I-Cache: Double Error Detection (DED), and

<table>
<thead>
<tr>
<th>RAM</th>
<th>Protection Scheme</th>
<th>Protection Granularity</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILM</td>
<td>ECC</td>
<td>64 bits</td>
<td>See Section 9.2</td>
</tr>
<tr>
<td>DLM</td>
<td>ECC</td>
<td>64 bits</td>
<td>See Section 9.2</td>
</tr>
<tr>
<td>I-Cache tag</td>
<td>Parity</td>
<td>8 bits</td>
<td>See Section 9.3</td>
</tr>
<tr>
<td>I-Cache data</td>
<td>Parity</td>
<td>8 bits</td>
<td>See Section 9.3</td>
</tr>
<tr>
<td>I-Cache tag</td>
<td>ECC</td>
<td>32 bits/64 bits</td>
<td>See Section 9.3</td>
</tr>
<tr>
<td>I-Cache data</td>
<td>ECC</td>
<td>32 bits</td>
<td>See Section 9.3</td>
</tr>
<tr>
<td>D-Cache tag</td>
<td>ECC</td>
<td>64 bits</td>
<td>See Section 9.4</td>
</tr>
<tr>
<td>D-Cache data</td>
<td>ECC</td>
<td>64 bits</td>
<td>See Section 9.4</td>
</tr>
<tr>
<td>BTB</td>
<td>none</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>STLB</td>
<td>none</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Note**

There are two possibilities of ECC Protection Granularity for I-Cache tag:

- 32 bits when SRAM data width is equal to or less than 39 bits, or
- 64 bits when SRAM data width is more than 39 bits.

9.1 Parity/ECC Control Mode

milmb.ECCEN, mdlmb.ECCEN, mcache_ctl.IC_ECCEN, and mcache_ctl.DC_ECCEN CSR fields control memory protection modes for ILM, DLM, I-Cache, and D-Cache respectively. The memory protection modes include:

- Disable parity/ECC.
• Generate exceptions on uncorrectable parity/ECC errors.
• Generate exceptions on all parity/ECC errors.

See the following subsections for detailed information on each mode.

9.1.1 Disable Parity/ECC

The behavior of the parity/ECC logic in the “Disable parity/ECC” mode is:
• The logic disables parity/ECC checking, but still updates the newly-generated parity/ECC code into the RAM for each write access.
• The received data is always uncorrected.
• The processor uses read-modify-writes to store a partial double word.
• If any parity/ECC error happens,
  – No exception will be generated.
  – The error will not be corrected.

9.1.2 Generate Exceptions on Uncorrectable Parity/ECC Errors

The behavior of the parity/ECC logic in the “Generate exceptions on uncorrectable parity/ECC errors” mode is:
• The logic enables parity/ECC checking and updates the newly-generated parity/ECC code into the RAM for each write access.
• Corrected data is always given.
• The processor uses read-modify-writes to store a partial double word.
• No exceptions will be generated if correctable errors are detected, and those correctable errors will be recovered.
• An exception will still be generated if an uncorrectable error is detected.
• For speculative accesses, e.g. instruction prefetch and data prefetch,
  – Uncorrectable errors does not generate exceptions
  – Correctable errors are corrected
9.1.3 Generate Exceptions on Parity/ECC Errors

The behavior of the parity/ECC logic in the “Generate exceptions on parity/ECC errors” mode is:

- The logic enables parity/ECC checking and updates the newly-generated parity/ECC code into the RAM for each write access.
- Corrected data is always given.
- The processor uses read-modify-writes to store a partial double word.
- For correctable Parity/ECC errors, exceptions are generated, and the corrupted data in the RAM will be corrected before generating exceptions.
- For uncorrectable Parity/ECC errors, exceptions are generated.
- For speculative accesses, e.g. instruction prefetch and data prefetch,
  - Uncorrectable and correctable errors does not generate exceptions
  - Correctable errors are corrected

9.2 Local Memory Protection

SECDED is applied to ILM and DLM when ILM or DLM Soft Error Protection is specified to ECC. One-bit ECC errors found on ILM/DLM access are correctable errors, while two-bit ECC errors found on load or store instructions on ILM/DLM are uncorrectable errors. For ILM/DLM ECC error exceptions, mecc_code will be updated with the ECC error information.

9.3 I-Cache Protection

SED and DED are applied to I-Cache when I-Cache Soft Error Protection is specified to Parity and ECC respectively. Data in I-Cache are always clean since there are no store accesses to I-Cache. If a parity or ECC error is found, I-Cache can get the correct copy from the next-level memory system. Therefore, if one-bit parity errors or one-bit/two-bit ECC errors are found on I-Cache accesses, they will be correctable errors. In order to check the soft-error on lock bit, another lock mirror bit is implemented. If a parity error occurs on locked cache line, or the lock bit and the lock mirror bit are mismatched, that parity error will be an uncorrectable error. For I-Cache parity error exceptions, mecc_code will be updated with the parity/ECC error information. For lock bit information, use the CCTL instruction, L1I_IX_RTAG, to get the lock bit value.
9.4 D-Cache Protection

SEC-DED is applied to D-Cache tag and data SRAMs when D-Cache Soft Error Protection is specified to ECC. An access to a cache line with a one-bit error on tag or data SRAMs is correctable, and the hardware will automatically repair the error. An access to a clean line with a two-bit error on data SRAMs is also correctable. The hardware will invalidate the line, and then fill the next-level memory system.

An access to a cache line with a two-bit error on tag SRAMs is uncorrectable. An access to a modified line with a two-bit error on data SRAMs is also uncorrectable. For D-Cache ECC error exceptions, mecc_code will be updated with the ECC error information.

9.5 Soft Error Injection

9.5.1 ILM/DLM ECC Error Injection

Use the following steps to inject ECC errors on ILM/DLM for implementing the ECC exception handler.

- Specify ILM/DLM to “Disable ECC” mode.
- Turn on milmb.RWECC or mdlmb.RWECC for ILM/DLM respectively.
- Perform a load instruction on an address in ILM/DLM region. Then, mecc_code.CODEx will contain the ECC code of the read data.
- Flip one or two bits in mecc_code.CODEx to inject one-bit or two-bit ECC errors.
- Perform a store instruction to the address in ILM/DLM region. Then, the ECC code with error will be written to ILM/DLM.
- Turn off milmb.RWECC or mdlmb.RWECC for ILM/DLM respectively.
- Specify ILM/DLM to “Generate Exceptions on Uncorrectable Parity/ECC Errors” mode or “Generate Exceptions on parity/ECC Errors” mode.
- Perform a load instruction to the address to trigger the ECC error.

9.5.2 I-Cache Parity/ECC Error Injection

Use the following steps to inject ECC errors on I-Cache for implementing the parity exception handler.

- Specify I-Cache to “Disable ECC” mode.
- Turn on mcachectl.IC_RWECC.
• Perform L1I_IX_RTAG or L1I_IX_RDATA CCTL command to a specific address to read parity data from tag RAM or data RAM respectively.

• Flip one bit in mecc_code.CODE or mcctldata to inject one bit parity error or flip the lock duplicate in mcctldata to generate a lock mismatch error.

• Perform L1I_IX_WTAG or L1I_IX_WDATA CCTL command to the specific address to inject a parity error.

• Turn off mcache_ctl.IC_RWECC.

• Specify I-Cache to “Generate Exceptions on Uncorrectable Parity/ECC Errors” mode or “Generate Exceptions on parity/ECC Errors” mode.

• Perform a fetch to that specific address to trigger the ECC error.

9.5.3 D-Cache ECC Error Injection

Use the following steps to inject ECC errors on D-Cache for implementing the ECC exception handler.

• Specify D-Cache to “Disable ECC” mode.

• Turn on mcache_ctl.DC_RWECC.

• Perform L1D_IX_RTAG or L1D_IX_RDATA CCTL command to a specific address to read ECC data from tag RAM or data RAM respectively.

• Flip one bit or two bits in mecc_code.CODE and mcctldata to inject one-bit or two-bit ECC errors.

• Perform L1D_IX_WTAG or L1D_IX_WDATA CCTL command to the specific address to inject an ECC error.

• Turn off mcache_ctl.DC_RWECC.

• Specify D-Cache to “Generate Exceptions on Uncorrectable Parity/ECC Errors” mode or “Generate Exceptions on parity/ECC Errors” mode.

• Perform load/store fetch to that specific address to trigger the ECC error.
10 Physical Memory Attributes

10.1 Introduction

Memory locations can have various attributes associated with them, and are basically categorized into either one of the two types: device region and memory region. While memory regions may be cacheable or non-cacheable locations, device regions are non-cacheable locations where accesses to these locations may cause side effects. AX45MP-1C uses MTYP (memory type) to define the cacheability and idempotency of memory regions. Please see descriptions of the MTYP field in Section 16.17.1 for possible values of MTYP.

AX45MP-1C provides two mechanisms for physical memory attributes:

- Static physical memory attributes
- Programmable physical memory attributes

10.2 Static Physical Memory Attributes

Up to 16 device regions could be statically configured in the processor through the Device Region configuration options. The memory type attributes for device regions are (device, non-bufferable).

Device regions and ILM/DLM should not overlap with each other. The behavior is UNDEFINED when they overlap.

10.3 Programmable Physical Memory Attributes

Programmable PMA allows dynamic adjustment of memory attributes in the runtime. It contains a configurable amount of PMA entries implemented as CSR registers to control the attributes of memory locations in interest. If the settings in those entries conflict with the static Device Region settings, PMA entries will have higher priorities.

PMA entries themselves are statically prioritized. The lowest-numbered PMA entry that matches any physical address (PA) of the access determines the attribute type and whether to support AMO instructions. If no PMA entries match the address, the attribute type is determined by the statically configured PMA. See Section 16.17.1 for more information about PMA entries.
10.4 Memory Access Ordering

Accesses to device regions are strongly-ordered. They are guaranteed to be non-speculative and issued in program order. An access to a device region is not issued until all preceding accesses to device regions are finished.

On the other hand, accesses to the memory regions could be speculative and the order of accessing memory regions is not guaranteed. A load access to a cacheable memory region might bypass an earlier store access if there is no data dependency. In such a scenario, explicit \texttt{FENCE} instructions are required to guarantee the order.

Table 23 shows ordering of two instructions A and B, where A < B (A comes earlier than B) in program order.

<table>
<thead>
<tr>
<th>A&lt;B in Program Order</th>
<th>B</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Cacheable</td>
<td>Non-</td>
<td>Device</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>Cacheable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Cacheable Memory</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Non-Cacheable Memory</td>
<td>-</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>Device</td>
<td>-</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

10.5 Non-Cacheable Memory and Device Accesses

Accessing to non-cacheable memory and device will bypass I-Cache, D-Cache and L2-Cache no matter the data is cached or not. The cache states are not changed by these accesses.

\textbf{Note}

For CPU revisions before 10.0.0, instruction fetch will access I-Cache when data in non-cacheable memory and device regions are cached into I-Cache. Under normal usages, data in non-cacheable memory and device regions are not cached unless PMA is changed from cacheable memory to non-cacheable memory or device regions.

10.6 Cacheable Memory Accesses to L1-Caches

This section describes operations to I-Cache and D-Cache.
10.6.1 Write-Back

Table 24: Behaviors upon Accessing Write-Back and No-Allocate Regions (MTYP=8)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L1</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L1</td>
</tr>
<tr>
<td></td>
<td>Change the line state to modified</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Write to L2</td>
</tr>
</tbody>
</table>

Table 25: Behaviors upon Accessing Write-Back and Read-Allocate Regions (MTYP=9)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L1</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L1</td>
</tr>
<tr>
<td></td>
<td>Change the line state to modified</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L2</td>
</tr>
<tr>
<td></td>
<td>Allocate the line in L1 with exclusive/shared state</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Write to L2</td>
</tr>
</tbody>
</table>

Table 26: Behaviors upon Accessing Write-Back and Write-Allocate Regions (MTYP=10)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L1</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L1</td>
</tr>
<tr>
<td></td>
<td>Change the line state to modified</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Allocate the line in L1 with modified state</td>
</tr>
<tr>
<td></td>
<td>Write to L1</td>
</tr>
</tbody>
</table>
Table 27: Behaviors upon Accessing Write-Back and Read-and-Write-Allocate Regions (MTYP=11)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L1</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L1</td>
</tr>
<tr>
<td></td>
<td>Change the line state to modified</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L2</td>
</tr>
<tr>
<td></td>
<td>Allocate the line in L1 with exclusive/shared state</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Allocate the line in L1 with modified state</td>
</tr>
<tr>
<td></td>
<td>Write to L1</td>
</tr>
</tbody>
</table>

When a write-miss happens for a write-no-allocate region (MTYP=8/9), the missed line is not allocated into D-Cache. The write data is written to L2. When Write-Around feature is configured, AX45MP-1C implements write buffers to improve write bandwidth to L2. See Section 7.4.4 for more details.

AX45MP-1C does not improve the read bandwidth of read-no-allocate region (MTYP=8/10). When a read-miss happens for read-no-allocate region (MTYP=8/10), D-Cache sends a single transfer to L2. The read to the same line is blocked until the transfer is completed.

10.6.2 I-Cache Disabled Behaviors

When `mcache_ctl.IC_EN` is 0, all fetches to cacheable memory regions are treated as non-cacheable and bufferable. For CPU revision before 2.0.0, the ARCACHE is not affected. For CPU revision 2.0.0 and later, ARCACHE is affected by `mcache_ctl.IC_EN`. When I-Cache is disabled, the requests are sent to MMIO port without passing through L2-Cache.

10.6.3 D-Cache Disabled Behaviors

When `mcache_ctl.DC_EN` is 0, all load and store instructions to cacheable memory regions are treated as non-cacheable and bufferable. When D-Cache is disabled, the requests are sent to MMIO port without passing through L2-Cache.

10.6.4 Debug Mode

In the debug mode, allocate attributes are ignored. AX45MP-1C does not allocate lines to D-Cache and I-Cache in the debug mode.
10.7 Cacheable Memory Accesses to L2-Caches

This section describes operations to L2-Cache. L2-Cache read requests include cache filling and non-allocate read. L2-Cache write requests include cache eviction and non-allocate write.

Table 28: Behaviors upon Accessing Write-Back and No-Allocate Regions (MTYP=8)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L2</td>
</tr>
<tr>
<td></td>
<td>Change the line state to dirty</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L3</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Write to L3</td>
</tr>
</tbody>
</table>

Table 29: Behaviors upon Accessing Write-Back and Read-Allocate Regions (MTYP=9)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L2</td>
</tr>
<tr>
<td></td>
<td>Change the line state to dirty</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L3</td>
</tr>
<tr>
<td></td>
<td>Allocate the line in L2 with clean state</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Write to L3</td>
</tr>
</tbody>
</table>

Table 30: Behaviors upon Accessing Write-Back and Write-Allocate Regions (MTYP=10)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L2</td>
</tr>
<tr>
<td></td>
<td>Change the line state to dirty</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L3</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Allocate the line in L2 with dirty state</td>
</tr>
<tr>
<td></td>
<td>Write to L2</td>
</tr>
</tbody>
</table>
Table 31: Behaviors upon Accessing Write-Back and Read-and-Write-Allocate Regions (MTYP=11)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Hit</td>
<td>Read from L2</td>
</tr>
<tr>
<td>Write-Hit</td>
<td>Write to L2</td>
</tr>
<tr>
<td></td>
<td>Change the line state to dirty</td>
</tr>
<tr>
<td>Read-Miss</td>
<td>Read from L3</td>
</tr>
<tr>
<td></td>
<td>Allocate the line in L2 with clean state</td>
</tr>
<tr>
<td>Write-Miss</td>
<td>Allocate the line in L2 with dirty state</td>
</tr>
<tr>
<td></td>
<td>Write to L2</td>
</tr>
</tbody>
</table>
11 MemBoost

11.1 Non-Blocking Memory Access

Non-blocking memory access hides memory latency by concurrently executing instructions and memory accesses. Store instructions in AX45MP-1C are always non-blocking. Load instructions in AX45MP-1C are blocking by default, meaning that they will block all subsequent instructions until the required bus accesses are finished (e.g., on cache misses, or when D-Cache is disabled or non-present).

The control bit mmisc_ctl.NBLD_EN controls the blocking behavior of load instructions. Once set, load instructions to cacheable memory will be non-blocking, waiting in the background for bus accesses to finish, and allow subsequent instructions to continue execution until data dependency hazards or structural hazards occur.

Data dependency hazards are hazards where subsequent instructions accessing the destination registers of the load instructions. Structural hazards are the resource limitations to allow load/store instructions to wait in the background. The number of cacheable accesses is limited by the maximum outstanding D-Cache misses.

When the processor operates in the non-blocking mode, the bus read transactions for the outstanding load/stores are each assigned a different AxID to allow the data to return out-of-order.

When load instructions operate in the blocking mode, bus aborts/errors are reported synchronously as Load Access Faults (mcause = 5). On the other hand, bus aborts/errors on load instructions will no longer be reported synchronously in the non-blocking mode. They will be treated as imprecise exceptions and reported as Bus Read/Write Transaction Error Local Interrupts (mip.BWEI).

Regardless of mmisc_ctl.NBLD_EN, the following memory access instructions are always blocking:

- Load instructions to device and non-cacheable memory
- Load instructions to local memory
- Load-Reserved/Store-Conditional instructions
- Atomic memory access (AMO) instructions
11.2 D-Cache Write-Around Support

Store operations to write-allocate memory will allocate a D-Cache entry on cache misses, in the hope that the entry will later be used again. However, in some streaming cases, such as memory copy or memory set operations for a large amount of data, where data is used once, allocating D-Cache entries is detrimental to performances. When Write-Around feature is configured, the store operations to the same write-no-allocate line are merged to a burst write transfer for improving performance. Write-no-allocate lines are memory regions with the following physical memory attributes:

- Cacheable memory, write-back, no allocate
- Cacheable memory, write-back, read-allocate
- D-Cache is in write-no-allocate mode (see below)
  - Cacheable memory, write-back, write-allocate
  - Cacheable memory, write-back, read and write-allocate

When \texttt{mcache_ctl.DC\_WAROUND} is set, the processor automatically detects streaming memory write patterns and makes D-Cache switch to write-no-allocate mode when the following conditions are met:

- A D-Cache miss happens for a store operation.
- This missing cache line is then completely filled by subsequent store operations. During this process, no cache miss for any other cache line happens for any store operation. Note that the same bytes in the missing cache line may be overwritten multiple times.
- The above mentioned conditions happen consecutively for a configured amount of cache lines. The corresponding addresses of those cache lines do not need to be consecutive.

In write-no-allocate mode, write-allocate attribute is overridden to write-no-allocate. For CPU revisions before 2.0.0, the overridden attribute affects D-Cache allocation policy, but AW\texttt{CACHE} of bus requests is not affected. For CPU revision 2.0.0 and later, both D-Cache allocation policy and AW\texttt{CACHE} are affected.

The processor stays in write-no-allocate mode until any of the following conditions happens:

- A D-Cache miss happens for a store operation while previous store operations have not completely filled their cache line.
- A load operation accesses the same cache line, which is being filled by store operations honored by Write-Around and is not full yet.
- \texttt{mcache_ctl.DC\_WAROUND} is changed.

The Write-Around feature should be configured and enabled (see Section 16.12.6) to take effect.
Note
• Stores qualified for the Write-Around behavior may produce AXI write transactions with partial or even zero write strobes in some corner case scenarios. As a result, Write-Around support should not be enabled for accessing regions containing designs that do not support AXI partial and zero write strobes.
• For CPU revision 10.0.0 and later, the processor does not produce AXI write transactions with zero write strobes.

11.3 D-Cache Prefetch

AX45MP-1C implements a hardware prefetcher for hiding memory access latency. The hardware prefetcher has a 4-entry reference prediction table, and each entry can detect an access sequence with a fixed stride. When \texttt{mcache\_ctl.DPREF\_EN} is set, the following access pattern enables the hardware prefetcher:

• A load instruction that accesses to a cacheable and read-allocate memory region
• The access causes cache misses
• The same load instruction accesses another two cache lines with a fixed stride (positive or negative).

After the pattern is detected, the hardware prefetcher automatically fills the next cache lines (with the same stride) into D-Cache. The hardware prefetcher does not send requests to system bus for the lines that have been in the D-Cache.

The prefetcher is stopped in one of the following conditions:

• The stride of the load access is changed
• Entering debug mode
• The load accesses
  – Cross LM boundary
  – Have inconsistent PMA
  – Cause access fault
  – Bus error
  – ECC error
  – PMP violations
  – PMA empty hole
  – Page fault
• SFENCE.VMA is executed
• SATP is updated
• Hit a load/store debug trigger
• A trap is taken
• A trap return instruction is executed
The prefetcher is also stopped when prefetch accesses encounter one of the following conditions. These errors will be ignored without triggering any exception.

- The accesses cross LM boundary
- The PMA is inconsistent with the load access
- Bus error
- ECC error
- PMP violations
- PMA empty hole
- Page fault


12 Coherence Manager (CM)

AX45MP-1C supports cache coherency of 1/2/4 cores with D-Cache. For 2-core and 4-core processors, cache coherency is always supported. For 1-core processors, cache coherency is supported only when IOCP is configured. Cache coherency is enabled only when mcache_ctl.DC_COHEN of an individual core is set. See Section 15.4 for enabling/disabling cache coherency.

Coherence manager implements MESI protocol to maintain coherency of cacheable accesses, and it supports up to 32 outstanding accesses. For handling requests, CM sends a request to L2-cache as well as probe messages to each core. If L2-Cache is not configured, the request is sent to memory bus interface. For a read request, CM sends a probe message to each core for writing back the modified line. For a write request, CM sends a probe message to each core for invalidating the line.
13 Trap

13.1 Introduction

According to the RISC-V Privileged Architecture, a trap is a control flow change of normal instruction execution caused by an interrupt or an exception. An interrupt is initiated by an external source, while an exception is generated as a by-product of instruction execution. When a trap happens, the processor stops processing the current flow of instructions, disables interrupts, saves enough states for later resumption, and starts executing a trap handler.

Interrupts can be local or external. The external interrupts are global interrupts that are arbitrated externally by a platform level interrupt controller (PLIC) and the selected external interrupt joins the rest of local interrupts for arbitration to take a trap.

Exceptions can be precise or imprecise. The instruction causing precise exceptions and all its subsequent instructions in the program order will not have affected the architectural state when precise exceptions are triggered. Furthermore, the events that cause these precise exceptions have to be precisely attributed to the causing instruction. The value of \textit{mcause} register will be greater than zero for precise exceptions. Exceptions not meeting these criteria can only be imprecise and they are delivered as local interrupts (\textit{mcause} < 0) instead. That is, the standard RISC-V privileged architecture exceptions are only triggered for precise exceptions, and local interrupts are triggered for imprecise exceptions.

For precise exceptions, \textit{mepc} is the PC of the faulting instruction. For imprecise exceptions, \textit{mepc} is pointing to the interrupted instruction. Regardless of preciseness of exceptions, \textit{mtval} records the effective faulting address for exceptions related to memory operations.

13.2 Interrupt

AX45MP-1C provides three interrupt inputs: timer interrupt, software interrupt, and external interrupt. Timer interrupts and software interrupts are local interrupts in a RISC-V platform, which means each processor in the platform receives its own timer/software interrupts. External interrupts are global interrupts in a RISC-V platform shared by all processors in a RISC-V platform. External interrupts are arbitrated and distributed by a platform-level interrupt controller (PLIC) to a processor. Each external interrupt source can be assigned its own priority, and each interrupt target (i.e., RISC-V processors) can select which external interrupt sources it will handle. PLIC routes the highest priority interrupt source to the target processor. See Section 18 for more descriptions on PLIC.
13.2.1 Additional Local Interrupts

In addition to external interrupts, AX45MP-1C may generate internal interrupts for the following events (imprecise exceptions):

- Local memory slave port parity/ECC error (Section 6.2. See mie.IMECCI and mip.IMECCI)
- Cache write back parity/ECC error (See mie.IMECCI and mip.IMECCI)
- Bus read/write transaction error (See mie.BWEI, mip.BWEI, and mdcause)
- Performance monitor overflow (See mie.PMOVI, mip.PMOVI, and mdcause)

13.2.2 Interrupt Status and Masking

The mip CSR contains pending bits of these interrupts, with the mie CSR contains enable bits of the respective interrupts. The processor can selectively enable interrupts by manipulating the mie CSR, or globally disable interrupts by clearing the mstatus.MIE bit.

13.3 Exception

AX45MP-1C implements the following (precise) exceptions (mcause > 0). See the tables in Section 16.3.13 (and Section 16.3.9) for how these exceptions can be identified by trap handlers.

- Instruction address misaligned exceptions
  - Jump to misaligned addresses
- Instruction access faults
  - Bus errors caused by instruction fetches
  - Uncorrectable ECC errors when fetching
  - PMP faults caused by instruction fetches
  - PMA faults caused by instruction fetches
- Illegal instructions
  - Unsupported instructions
  - Privileged instructions
  - Accessing non-existent CSRs
  - Accessing privileged CSRs
  - Writing to read-only CSRs
  - Executing Andes-specific instructions in the RISC-V compatibility mode (mmisc_ctl.RVCOMPM == 1).
  - Executing floating point instructions without enabling FPU context (mstatus.FS == 0).
- Breakpoint exceptions
- Load address misaligned exceptions
• Load access faults
  – Bus errors caused by load instructions
  – ECC errors caused by load instructions
  – PMP fault caused by load instructions
  – PMA fault caused by load instructions
• Store/AMO address misaligned exceptions
• Store/AMO access faults
  – Bus errors caused by store instructions
  – ECC errors caused by store instructions
  – PMP fault caused by store instructions
  – PMA fault caused by store instructions
• Environment calls
• Stack overflow/underflow exceptions with StackSafe supported

Some events (for example, parity/ECC and bus errors) in this list may cause imprecise exceptions in some circumstances instead. Imprecise exceptions are delivered through local interrupts (\(mcause < 0\)) instead of the standard RISC-V exceptions (\(mcause > 0\)). It all depends on the ability of the pipeline to attribute the errors to the faulting instruction and keep the architectural state clean from being polluted by the faulting instruction and all of its subsequent instructions. For example, bus read errors on non-critical word cannot be attributed to any of the executed instructions and will be imprecise. As another example, when the processor is in the non-blocking mode (Section 11.1), load instructions could have been retired before data returns and bus errors will always be imprecise.

Most of errors related to address checks are precise, unless the instruction is split into micro-operations and the error is found not on the first micro-operation. For example, PMP checks errors for the second micro-operation of a misaligned memory accesses.

13.4 Trap Handling

13.4.1 Entering the Trap Handler

When a trap occurs, the following operations are applied:

• \(mepc\) is set to the current program counter.
• \(mstatus\) is updated.
  – The \(MPP\) field is set to the current privilege mode.
  – The \(MPIE\) field is set to the \(MIE\) field.
  – The \(MIE\) field is set to 0.
• \(mcause\) is updated.
• \(mtval\) is updated on any of address-misaligned, access-fault, or page-fault exceptions.
• The privilege mode is changed to M-mode.
• When mmisc_ctl.VEC_PLIC is 0, the program counter is set to the address specified by mtvec.
• When mmisc_ctl.VEC_PLIC is 1, the mtvec register will be the base address register of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.
  – mtvec[0] is for exceptions and non-external local interrupts. For these traps, the mcause register records the trap type based on RISC-V definitions.
  – mtvec[i] is for external PLIC interrupt source i triggered through the mip.MEIP pending condition.
  – mtvec[1024+i] is for external PLIC interrupt source i triggered through
    * the mip.SEIP pending condition when mideleg.SEI == 0 for M/S/U systems.
    * the mip.UEIP pending condition when mideleg.UEI == 0 for M/U systems.
  – mtvec[2048+i] is for external PLIC interrupt source i triggered through the mip.UEIP pending condition when mideleg.UEI == 0 for M/S/U systems.
  – For external PLIC interrupts, the mcause register records the interrupt source ID. The RISC-V architecture defines a two-level stack of interrupt enable bits and privilege modes. To support nested traps, the trap handler should back up trap handling CSRs and enable the interrupt enable bit.

13.4.2 Returning from the Trap Handler

After handling a trap, the MRET instruction can be executed for returning to the instruction and the privilege context before the trap happened. Alternatively, the trap handler could assign new PC, privilege level and/or interrupt enable status to mepc, mstatus.MPP and mstatus.MPIE before MRET. Specifically, the following operations take place when an MRET instruction is executed:

• The program counter is set to mepc
• The privilege mode is set to mstatus.MPP
• mstatus is updated
  – The MPP field is set to U-mode (or M-mode if U-mode is not supported)
  – The MIE field is set to the MPIE field
  – The MPIE field is set to 1
14 Reset and Non-Maskable Interrupts

14.1 Reset

When the processor is out of reset, the following operations are applied:

- CSRs are set to their reset values.
- All integer registers (listed in Table 1) are set to zero.
- BTB is initialized.
- Program execution starts at the reset vector specified in the mnvec CSR.

14.2 Non-Maskable Interrupts

Non-maskable interrupts (NMIs) are intended for handling hardware error conditions and are assumed to be non-resumable. They are triggered through the NMI input signal.

The following operations are applied when an NMI is taken:

- The mepc register is written with the address of the next instruction when the NMI was taken.
- The mcause register is set to 1, indicating that NMI is caused by the reset vector specified in the mnvec SCR.
- The mstatus.MPP field records the privilege mode before NMI was taken.
- The mstatus.MPIE field is set to the value of mstatus.xIE before NMI was taken. The “x” is the active privilege mode before the NMI was taken.
- The mstatus.MIE field is set to 0.
15 Power Management

15.1 Power Management Unit

A power management unit is used to control clocks, resets, isolation cells and power switches. AX45MP-1C does not provide the power management unit.

15.2 Wait-For-Interrupt Mode

The processor enters the wait-for-interrupt (WFI) mode with the \texttt{WFI} instruction for reducing power consumption, and clock-gating or power-gating of the processor should only happen when the processor is in the WFI mode.

Once in the WFI mode, memory transactions that are started before the execution of \texttt{WFI} are guaranteed to have been completed. All transient states of memory handling are flushed, and no new memory accesses will take place.

In this period, the core and bus clocks can be safely gated to reduce the power consumption or changed for frequency scaling. This is also the safe period to power-gate the processor and leave the I/D-Cache SRAMs entering the state retention mode.

The availability just depends on whether the LM clock is active. If slave port accesses are still needed in WFI mode, the LM clock should still be clocked while the core clock may be gated off.

NMI, debug and other interrupts defined in the \texttt{mip} CSR may cause the processor to leave the WFI mode.

The NMI or debug interrupt cause the processor to leave the WFI mode unconditionally. The processor will resume and start to execute from the first instruction of NMI or debug-interrupt service routine.

All interrupts defined in the \texttt{mip} CSR may cause the processor to leave the WFI mode, depending on the setting of the \texttt{mie} CSR: interrupts disabled by the \texttt{mie} CSR cannot wake up the processor. However, the processor can be awoken by these interrupts regardless the value of the global interrupt enable bit (\texttt{mstatus.MIE}).

When the processor is awoken by a pending interrupt and \texttt{mstatus.MIE} is enabled, it will resume and start to execute from the corresponding interrupt service routine. When the processor is awoken by a pending interrupt and \texttt{mstatus.MIE} is disabled, it will resume and start to execute from the instruction after the WFI instruction.
Please note that the RISC-V ISA only defines the `WFI` instruction as a hint instruction. For portability, `WFI` instructions should not be assumed to always cause the processor to pause until an interrupt arrives. They may be implemented as NOPs by other implementations and should be inside loops that stop when `mie` & `mip` are not zero.

### 15.3 Dynamic Frequency Scaling (DFS)

When the Core Interface is “asynchronous”, the core clocks are asynchronous to L2 clock and bus clock. An external clock generator can be implemented for scaling the core clocks.

**Note**

Andes does not provide the external clock generator. The clock generator should generate stable clocks.

### 15.4 D-Cache Coherency

AX45MP-1C supports cache coherency with D-Cache and coherence manager (CM). When a core is in WFI mode, CM might still send probe requests to the core. When the core receives probe requests, the WFI mode of the core is still asserted. For responding the probe requests, the dcache clock of the core should be enabled. Before disabling all clocks of a core or powering down the core, D-Cache coherency should be disabled.

#### 15.4.1 Disable D-Cache Coherency

To disable D-Cache coherency of a core, apply the following sequence:

- Disable D-Cache by clearing `mcache_ctl.DC_EN`. This step changes accesses to non-cacheable regions for preventing D-Cache from allocating more lines.
- Write back and invalidate D-Cache by executing the CCTL command `L1D_WBINVAL_ALL`. All modified lines are flushed and D-Cache will not cache stale lines after power-up.
- Disable D-Cache coherency by clearing `mcache_ctl.DC_COHEN`. CM will not send probe requests while D-Cache is powered down.
- Wait for `mcache_ctl.DC_COHSTA` to be cleared to ensure the previous step is completed.

#### 15.4.2 Enable D-Cache Coherency

To enable coherency of the core, apply the following sequence:

- Enable D-Cache coherency by setting `mcache_ctl.DC_COHEN`. 
• Wait for mcache_ctl.DC_COHSTA to be set.
• Enable D-Cache by setting mcache_ctl.DC_EN.

For maintaining a coherent view between cores, D-Cache coherency should be enabled before D-Cache is enabled.

15.5 Disable All Clocks of a Core

To disable all clocks of a core, apply the following sequence:

• Disable D-Cache coherency by applying the sequence in Section 15.4.1.
• Execute the WFI instruction and wait for the processor to enter WFI mode.
• Clocks should not be disabled until the processor enters WFI mode.
16 Control and Status Registers

16.1 Introduction

The sections below describe the registers in detail.

16.1.1 System Register Type

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>Implementation dependent/determined</td>
</tr>
<tr>
<td>RO</td>
<td>Read-Only register/field. Any software write to RO register/field will be silently ignored by hardware.</td>
</tr>
<tr>
<td>RW</td>
<td>Read/Write register/field</td>
</tr>
<tr>
<td>W1</td>
<td>Write-only. Only writing 1 has an effect.</td>
</tr>
<tr>
<td>W1S</td>
<td>Write 1 to Set</td>
</tr>
<tr>
<td>W1C</td>
<td>Write 1 to Clear</td>
</tr>
<tr>
<td>WLRL</td>
<td>Write/Read Only Legal Values</td>
</tr>
<tr>
<td>WARL</td>
<td>Write-Any-Read-Legal</td>
</tr>
</tbody>
</table>

16.1.2 Reset Value

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>The reset value is “Don’t Care”</td>
</tr>
</tbody>
</table>

16.1.3 CSR Listing

Table 32: Machine Information Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvendorid</td>
<td>0xf11</td>
<td>Section 16.2.1</td>
</tr>
<tr>
<td>marchid</td>
<td>0xf12</td>
<td>Section 16.2.2</td>
</tr>
<tr>
<td>mimpid</td>
<td>0xf13</td>
<td>Section 16.2.3</td>
</tr>
<tr>
<td>mhartid</td>
<td>0xf14</td>
<td>Section 16.2.4</td>
</tr>
<tr>
<td>Mnemonic Name</td>
<td>CSR Address</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>mstatus</td>
<td>0x300</td>
<td>Section 16.3.1</td>
</tr>
<tr>
<td>misa</td>
<td>0x301</td>
<td>Section 16.3.2</td>
</tr>
<tr>
<td>medeleg</td>
<td>0x302</td>
<td>Section 16.3.3</td>
</tr>
<tr>
<td>mideleg</td>
<td>0x303</td>
<td>Section 16.3.4</td>
</tr>
<tr>
<td>mie</td>
<td>0x304</td>
<td>Section 16.3.5</td>
</tr>
<tr>
<td>mtvec</td>
<td>0x305</td>
<td>Section 16.3.6</td>
</tr>
<tr>
<td>mscratch</td>
<td>0x340</td>
<td>Section 16.3.7</td>
</tr>
<tr>
<td>mepc</td>
<td>0x341</td>
<td>Section 16.3.8</td>
</tr>
<tr>
<td>mcause</td>
<td>0x342</td>
<td>Section 16.3.9</td>
</tr>
<tr>
<td>mtval</td>
<td>0x343</td>
<td>Section 16.3.10</td>
</tr>
<tr>
<td>mip</td>
<td>0x344</td>
<td>Section 16.3.11</td>
</tr>
<tr>
<td>mxstatus</td>
<td>0x7c4</td>
<td>Section 16.3.12</td>
</tr>
<tr>
<td>mdcause</td>
<td>0x7c9</td>
<td>Section 16.3.13</td>
</tr>
<tr>
<td>mslideleg</td>
<td>0x7D5</td>
<td>Section 16.3.14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcycle</td>
<td>0xb00</td>
<td>Section 16.4.1</td>
</tr>
<tr>
<td>minstret</td>
<td>0xb02</td>
<td>Section 16.4.2</td>
</tr>
<tr>
<td>mhpmcnter3</td>
<td>0xb03</td>
<td>Section 16.4.3</td>
</tr>
<tr>
<td>mhpmcnter4</td>
<td>0xb04</td>
<td>Section 16.4.3</td>
</tr>
<tr>
<td>mhpmcnter5</td>
<td>0xb05</td>
<td>Section 16.4.3</td>
</tr>
<tr>
<td>mhpmcnter6</td>
<td>0xb06</td>
<td>Section 16.4.3</td>
</tr>
<tr>
<td>mcounteren</td>
<td>0x306</td>
<td>Section 16.4.6</td>
</tr>
<tr>
<td>mhpmevent3</td>
<td>0x323</td>
<td>Section 16.4.5</td>
</tr>
<tr>
<td>mhpmevent4</td>
<td>0x324</td>
<td>Section 16.4.5</td>
</tr>
<tr>
<td>mhpmevent5</td>
<td>0x325</td>
<td>Section 16.4.5</td>
</tr>
<tr>
<td>mhpmevent6</td>
<td>0x326</td>
<td>Section 16.4.5</td>
</tr>
<tr>
<td>mcountinhibit</td>
<td>0x320</td>
<td>Section 16.4.4</td>
</tr>
<tr>
<td>mcounterwen</td>
<td>0x7ce</td>
<td>Section 16.4.7</td>
</tr>
<tr>
<td>mcounterinten</td>
<td>0x7cf</td>
<td>Section 16.4.8</td>
</tr>
<tr>
<td>mcountermask_m</td>
<td>0x7d1</td>
<td>Section 16.4.9</td>
</tr>
<tr>
<td>mcountermask_s</td>
<td>0x7d2</td>
<td>Section 16.4.10</td>
</tr>
</tbody>
</table>
### Table 34: (continued)

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcountermask_u</td>
<td>0x7d3</td>
<td>Section 16.4.11</td>
</tr>
<tr>
<td>mcounterovf</td>
<td>0x7d4</td>
<td>Section 16.4.12</td>
</tr>
</tbody>
</table>

### Table 35: Configuration Control & Status Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>micmCfg</td>
<td>0xfc0</td>
<td>Section 16.5.1</td>
</tr>
<tr>
<td>mdcmCfg</td>
<td>0xfc1</td>
<td>Section 16.5.2</td>
</tr>
<tr>
<td>mmScCfg</td>
<td>0xfc2</td>
<td>Section 16.5.3</td>
</tr>
</tbody>
</table>

### Table 36: Trigger Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>tSelect</td>
<td>0x7a0</td>
<td>Section 16.6.1</td>
</tr>
<tr>
<td>tData1</td>
<td>0x7a1</td>
<td>Section 16.6.2</td>
</tr>
<tr>
<td>tData2</td>
<td>0x7a2</td>
<td>Section 16.6.3</td>
</tr>
<tr>
<td>tData3</td>
<td>0x7a3</td>
<td>Section 16.6.4</td>
</tr>
<tr>
<td>tInfo</td>
<td>0x7a4</td>
<td>Section 16.6.5</td>
</tr>
<tr>
<td>tControl</td>
<td>0x7a5</td>
<td>Section 16.6.6</td>
</tr>
<tr>
<td>mContext</td>
<td>0x7a8</td>
<td>Section 16.6.7</td>
</tr>
<tr>
<td>sContext</td>
<td>0x7aa</td>
<td>Section 16.6.8</td>
</tr>
<tr>
<td>mControl</td>
<td>0x7a1</td>
<td>Section 16.6.9</td>
</tr>
<tr>
<td>iCount</td>
<td>0x7a1</td>
<td>Section 16.6.10</td>
</tr>
<tr>
<td>iTrigger</td>
<td>0x7a1</td>
<td>Section 16.6.11</td>
</tr>
<tr>
<td>eTrigger</td>
<td>0x7a1</td>
<td>Section 16.6.12</td>
</tr>
<tr>
<td>textra</td>
<td>0x7a3</td>
<td>Section 16.6.13</td>
</tr>
</tbody>
</table>

### Table 37: Debug Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>dCsr</td>
<td>0x7b0</td>
<td>Section 16.7.1</td>
</tr>
<tr>
<td>dPc</td>
<td>0x7b1</td>
<td>Section 16.7.2</td>
</tr>
<tr>
<td>dScratch0</td>
<td>0x7b2</td>
<td>Section 16.7.3</td>
</tr>
<tr>
<td>dScratch1</td>
<td>0x7b3</td>
<td>Section 16.7.4</td>
</tr>
<tr>
<td>dExc2dbg</td>
<td>0x7e0</td>
<td>Section 16.7.5</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcause</td>
<td>0x7e1</td>
<td>Section 16.7.6</td>
</tr>
</tbody>
</table>

Table 38: Supervisor Trap Related Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sstatus</td>
<td>0x100</td>
<td>Section 16.8.1</td>
</tr>
<tr>
<td>sedeleq</td>
<td>0x102</td>
<td>Section 16.8.2</td>
</tr>
<tr>
<td>sdeleg</td>
<td>0x103</td>
<td>Section 16.8.3</td>
</tr>
<tr>
<td>sie</td>
<td>0x104</td>
<td>Section 16.8.4</td>
</tr>
<tr>
<td>stvec</td>
<td>0x105</td>
<td>Section 16.8.5</td>
</tr>
<tr>
<td>scounteren</td>
<td>0x106</td>
<td>Section 16.8.6</td>
</tr>
<tr>
<td>sscratch</td>
<td>0x140</td>
<td>Section 16.8.7</td>
</tr>
<tr>
<td>sepc</td>
<td>0x141</td>
<td>Section 16.8.8</td>
</tr>
<tr>
<td>scause</td>
<td>0x142</td>
<td>Section 16.8.9</td>
</tr>
<tr>
<td>stval</td>
<td>0x143</td>
<td>Section 16.8.10</td>
</tr>
<tr>
<td>sip</td>
<td>0x144</td>
<td>Section 16.8.11</td>
</tr>
<tr>
<td>slie</td>
<td>0x9c4</td>
<td>Section 16.8.12</td>
</tr>
<tr>
<td>slip</td>
<td>0x9c5</td>
<td>Section 16.8.13</td>
</tr>
<tr>
<td>sdcause</td>
<td>0x9c9</td>
<td>Section 16.8.14</td>
</tr>
</tbody>
</table>

Table 39: Supervisor Page Translation Related Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>satp</td>
<td>0x180</td>
<td>Section 16.9.1</td>
</tr>
</tbody>
</table>

Table 40: Supervisor Counter Related Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>scountermask_m</td>
<td>0x9d1</td>
<td>Section 16.10.1</td>
</tr>
<tr>
<td>scountermask_s</td>
<td>0x9d2</td>
<td>Section 16.10.2</td>
</tr>
<tr>
<td>scountermask_u</td>
<td>0x9d3</td>
<td>Section 16.10.3</td>
</tr>
<tr>
<td>scounterinten</td>
<td>0x9cf</td>
<td>Section 16.10.4</td>
</tr>
<tr>
<td>scounterovf</td>
<td>0x9d4</td>
<td>Section 16.10.5</td>
</tr>
<tr>
<td>scountinhbit</td>
<td>0x9e0</td>
<td>Section 16.10.6</td>
</tr>
<tr>
<td>shpmvent3</td>
<td>0x9E3</td>
<td>Section 16.10.7</td>
</tr>
</tbody>
</table>

Continued on next page…
Table 40: (continued)

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>shpmevent4</td>
<td>0x9E4</td>
<td>Section 16.10.7</td>
</tr>
<tr>
<td>shpmevent5</td>
<td>0x9E5</td>
<td>Section 16.10.7</td>
</tr>
<tr>
<td>shpmevent6</td>
<td>0x9E6</td>
<td>Section 16.10.7</td>
</tr>
</tbody>
</table>

Table 41: User Trap Related Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ustatus</td>
<td>0x000</td>
<td>Section 16.11.1</td>
</tr>
<tr>
<td>uie</td>
<td>0x004</td>
<td>Section 16.11.2</td>
</tr>
<tr>
<td>utvec</td>
<td>0x005</td>
<td>Section 16.11.3</td>
</tr>
<tr>
<td>uscratch</td>
<td>0x040</td>
<td>Section 16.11.4</td>
</tr>
<tr>
<td>uepc</td>
<td>0x041</td>
<td>Section 16.11.5</td>
</tr>
<tr>
<td>ucause</td>
<td>0x042</td>
<td>Section 16.11.6</td>
</tr>
<tr>
<td>utval</td>
<td>0x043</td>
<td>Section 16.11.7</td>
</tr>
<tr>
<td>uip</td>
<td>0x044</td>
<td>Section 16.11.8</td>
</tr>
<tr>
<td>udcause</td>
<td>0x809</td>
<td>Section 16.11.9</td>
</tr>
</tbody>
</table>

Table 42: Memory and Miscellaneous Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>milmb</td>
<td>0x7c0</td>
<td>Section 16.12.1</td>
</tr>
<tr>
<td>mdlimb</td>
<td>0x7c1</td>
<td>Section 16.12.2</td>
</tr>
<tr>
<td>mecc_code</td>
<td>0x7c2</td>
<td>Section 16.12.3</td>
</tr>
<tr>
<td>mnvec</td>
<td>0x7c3</td>
<td>Section 16.12.4</td>
</tr>
<tr>
<td>mpt_ctl</td>
<td>0x7c5</td>
<td>Section 16.12.5</td>
</tr>
<tr>
<td>mcache_ctl</td>
<td>0x7ca</td>
<td>Section 16.12.6</td>
</tr>
<tr>
<td>mcctlbeginaddr</td>
<td>0x7cb</td>
<td>Section 16.12.9</td>
</tr>
<tr>
<td>mcctlcommand</td>
<td>0x7cc</td>
<td>Section 16.12.10</td>
</tr>
<tr>
<td>mcctldata</td>
<td>0x7cd</td>
<td>Section 16.12.11</td>
</tr>
<tr>
<td>scctldata</td>
<td>0x9cd</td>
<td>Section 16.12.12</td>
</tr>
<tr>
<td>ucctlbeginaddr</td>
<td>0x80b</td>
<td>Section 16.12.13</td>
</tr>
<tr>
<td>ucctlcommand</td>
<td>0x80c</td>
<td>Section 16.12.14</td>
</tr>
<tr>
<td>mmisc_ctl</td>
<td>0x7d0</td>
<td>Section 16.12.7</td>
</tr>
<tr>
<td>mclk_ctl</td>
<td>0x7df</td>
<td>Section 16.12.8</td>
</tr>
</tbody>
</table>
### Table 43: Hardware Stack Protection and Recording Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mhsp_ctl</td>
<td>0x7c6</td>
<td>Section 16.13.1</td>
</tr>
<tr>
<td>msp_bound</td>
<td>0x7c7</td>
<td>Section 16.13.2</td>
</tr>
<tr>
<td>msp_base</td>
<td>0x7c8</td>
<td>Section 16.13.3</td>
</tr>
</tbody>
</table>

### Table 44: CoDense Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>uitb</td>
<td>0x800</td>
<td>Section 16.14.1</td>
</tr>
</tbody>
</table>

### Table 45: DSP Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucode</td>
<td>0x801</td>
<td>Section 16.15.1</td>
</tr>
</tbody>
</table>

### Table 46: PMP Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmpcfg0</td>
<td>0x3a0</td>
<td>Section 16.16.1</td>
</tr>
<tr>
<td>pmpcfg2</td>
<td>0x3a2</td>
<td>Section 16.16.1</td>
</tr>
<tr>
<td>pmpaddr0</td>
<td>0x3b0</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr1</td>
<td>0x3b1</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr2</td>
<td>0x3b2</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr3</td>
<td>0x3b3</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr4</td>
<td>0x3b4</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr5</td>
<td>0x3b5</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr6</td>
<td>0x3b6</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr7</td>
<td>0x3b7</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr8</td>
<td>0x3b8</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr9</td>
<td>0x3b9</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr10</td>
<td>0x3ba</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr11</td>
<td>0x3bb</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr12</td>
<td>0x3bc</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr13</td>
<td>0x3bd</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr14</td>
<td>0x3be</td>
<td>Section 16.16.2</td>
</tr>
<tr>
<td>pmpaddr15</td>
<td>0x3bf</td>
<td>Section 16.16.2</td>
</tr>
</tbody>
</table>
### Table 47: PMA Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmacfg0</td>
<td>0xbc0</td>
<td>Section 16.17.1</td>
</tr>
<tr>
<td>pmacfg2</td>
<td>0xbc2</td>
<td>Section 16.17.1</td>
</tr>
<tr>
<td>pmaaddr0</td>
<td>0xbd0</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr1</td>
<td>0xbd1</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr2</td>
<td>0xbd2</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr3</td>
<td>0xbd3</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr4</td>
<td>0xbd4</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr5</td>
<td>0xbd5</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr6</td>
<td>0xbd6</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr7</td>
<td>0xbd7</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr8</td>
<td>0xbd8</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr9</td>
<td>0xbd9</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr10</td>
<td>0xbd10</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr11</td>
<td>0xbd11</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr12</td>
<td>0xbd12</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr13</td>
<td>0xbd13</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr14</td>
<td>0xbd14</td>
<td>Section 16.17.2</td>
</tr>
<tr>
<td>pmaaddr15</td>
<td>0xbd15</td>
<td>Section 16.17.2</td>
</tr>
</tbody>
</table>

### Table 48: Floating-Point CSRs

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>fflags</td>
<td>0x001</td>
<td>Section 16.18.1</td>
</tr>
<tr>
<td>frm</td>
<td>0x002</td>
<td>Section 16.18.2</td>
</tr>
<tr>
<td>fcsr</td>
<td>0x003</td>
<td>Section 16.18.3</td>
</tr>
</tbody>
</table>

### Table 49: User Mode Counter Related Registers

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycle</td>
<td>0xc00</td>
<td>Section 16.19.1</td>
</tr>
<tr>
<td>time</td>
<td>0xc01</td>
<td>Section 16.19.2</td>
</tr>
<tr>
<td>instret</td>
<td>0xc02</td>
<td>Section 16.19.3</td>
</tr>
<tr>
<td>hpmcounter3</td>
<td>0xc03</td>
<td>Section 16.19.4</td>
</tr>
<tr>
<td>hpmcounter4</td>
<td>0xc04</td>
<td>Section 16.19.4</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 49: (continued)

<table>
<thead>
<tr>
<th>Mnemonic Name</th>
<th>CSR Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpmcounter5</td>
<td>0xc05</td>
<td>Section 16.19.4</td>
</tr>
<tr>
<td>hpmcounter6</td>
<td>0xc06</td>
<td>Section 16.19.4</td>
</tr>
</tbody>
</table>
### 16.2 Machine Information Registers

#### 16.2.1 Machine Vendor ID Register

*Mnemonic Name:* mvendorid  
*IM Requirement:* Required  
*Access Mode:* Machine  
*CSR Address:* 0xf11 (standard read only)

This read-only register provides the Andes JEDEC manufacturer ID: 0x0000031e.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVENDORID</td>
<td>[63:0]</td>
<td>The manufacturer ID of Andes</td>
<td>RO</td>
<td>0x0000031e</td>
</tr>
</tbody>
</table>

#### 16.2.2 Machine Architecture ID Register

*Mnemonic Name:* marchid  
*IM Requirement:* Required  
*Access Mode:* Machine  
*CSR Address:* 0xf12 (standard read only)

This register provides the micro-architecture id of AndesCore processor implementations. For AX45MP-1C, `marchid.CPU_ID` will be 0x8a45. Note that the MSB of this register is 1 for commercial implementations of RISC-V processors.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_ID</td>
<td>[30:0]</td>
<td>Andes CPU ID</td>
<td>RO</td>
<td>0x8a45</td>
</tr>
</tbody>
</table>
16.2.3 Machine Implementation ID Register

Mnemonic Name: mimpid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf13 (standard read only)

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0   | 7   | 4   | 3   | 2   | 1   | 0   | 31  |
| MAJOR| MINOR| EXTENSION |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

This register is used to identify the revision number of the processor, which is denoted using the MAJOR.MINOR.EXTENSION format.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTENSION</td>
<td>[3:0]</td>
<td>Revision extension</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td>MINOR</td>
<td>[7:4]</td>
<td>Revision minor</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td>MAJOR</td>
<td>[31:8]</td>
<td>Revision major</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>
16.2.4 Hart ID Register

Mnemonic Name: mhartid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf14 (standard read only)

This register provides the ID of the hardware thread. It is required that one of the hart IDs must be zero on a RISC-V platform.

16.3 Machine Trap Related CSRs

16.3.1 Machine Status

Mnemonic Name: mstatus
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x300 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIE</td>
<td>[0]</td>
<td>U-mode interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SIE</td>
<td>[1]</td>
<td>S-mode interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Value | Meaning  
--- | ---
0   | Disabled
1   | Enabled

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>UPIE</td>
<td>[4]</td>
<td>UPIE holds the value of the UIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SPIE</td>
<td>[5]</td>
<td>SPIE holds the value of the SIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MPIE</td>
<td>[7]</td>
<td>MPIE holds the value of the MIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SPP</td>
<td>[8]</td>
<td>SPP holds the privilege mode prior to a trap. Encoding is 1 for S-mode and 0 for U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MPP</td>
<td>[12:11]</td>
<td>MPP holds the privilege mode prior to a trap. When U-mode is not available, this field is hardwired to 3.</td>
<td>WARL</td>
<td>3</td>
</tr>
</tbody>
</table>

Continued on next page...
Field Name | Bits | Description | Type | Reset
---|---|---|---|---
FS | [14:13] | FS holds the status of the architectural states of the floating-point unit, including the fcsr CSR and f0 – f31 floating-point data registers. The value of this field is zero and read-only if the processor does not have FPU. This field is primarily managed by software. The processor hardware assists the state managements in two regards:
• Attempts to access fcsr or any f register raise an illegal-instruction exception when FS is Off.
• FS is updated to the Dirty state with the execution of any instruction that updates fcsr or any f register when FS is Initial or Clean.
Changing the setting of this field has no effect on the contents of the floating-point register states. In particular, setting FS to Off does not destroy the states, nor does setting FS to Initial clear the contents.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Initial</td>
</tr>
<tr>
<td>2</td>
<td>Clean</td>
</tr>
<tr>
<td>3</td>
<td>Dirty</td>
</tr>
</tbody>
</table>

MPRV | [17] | When the MPRV bit is set, the memory access privilege for load and store are specified by the MPP field. When U-mode is not available, this field is hardwired to 0. | RW | 0

Continued on next page...
### Field Name | Bits | Description | Type | Reset
--- | --- | --- | --- | ---
SUM | [18] | SUM controls whether a S-mode load/store instruction to a user accessible page is allowed or not when page translation is enabled. It is in effect in two scenarios: (a) M-mode with MPRV=1 and MP\(P\)=S, and (b) in S-mode. It has no effect when page-based virtual memory is not in effect. A page is user accessible when the U bit of the corresponding PTE entry is 1. It is hardwired to 0 when S-mode is not supported. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>1</td>
<td>Allowed</td>
</tr>
</tbody>
</table>

MXR | [19] | MXR controls whether execute-only pages are readable. It has no effect when page-based virtual memory is not in effect. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execute-only pages are not readable</td>
</tr>
<tr>
<td>1</td>
<td>Execute-only pages are readable</td>
</tr>
</tbody>
</table>

TVM | [20] | TVM controls whether performing certain virtual memory operations in S-mode will raise illegal instruction exceptions. The operations include accessing the satp register and executing the SFENCE.VMA instruction. It is hardwired to 0 when S-mode is not supported. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal execution</td>
</tr>
<tr>
<td>1</td>
<td>Raising exceptions</td>
</tr>
</tbody>
</table>

Continued on next page...
TW [21]  
TW controls whether executing WFI instructions in S-mode will raise illegal instruction exceptions. It is hardwired to 0 when S-mode is not supported.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal execution</td>
</tr>
<tr>
<td>1</td>
<td>Raising exceptions</td>
</tr>
</tbody>
</table>

TSR [22]  
TSR controls whether executing SRET instructions in S-mode will raise illegal instruction exceptions. It is hardwired to 0 when S-mode is not supported.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal execution</td>
</tr>
<tr>
<td>1</td>
<td>Raising exceptions</td>
</tr>
</tbody>
</table>

UXL [33:32]  
UXL controls the value of XLEN for U-mode. When U-mode is not available, this field is hardwired to 0.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
</tbody>
</table>

SXL [35:34]  
SXL controls the value of XLEN for S-mode. When S-mode is not available, this field is hardwired to 0.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
</tbody>
</table>

SD [63]  
SD summarizes whether either the FS field or XS field is dirty.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When supervisor mode or N extension is not supported, the corresponding bits in mstatus are hardwired to zero.
16.3.2 Machine ISA Register

Mnemonic Name: misa
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x301 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[0]</td>
<td>Atomic extension</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>[1]</td>
<td>Tentatively reserved for Bit operations extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>[2]</td>
<td>Compressed extension</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>[4]</td>
<td>RV32E base ISA</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>[6]</td>
<td>Additional standard extensions present</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>[7]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>[8]</td>
<td>RV32I/64I/128I base ISA</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td>[9]</td>
<td>Tentatively reserved for Dynamically Translated Languages extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>[10]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>[11]</td>
<td>Tentatively reserved for Decimal Floating-Point extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>[12]</td>
<td>Integer Multiply/Divide extension</td>
<td>RO</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>[13]</td>
<td>User-level interrupts supported</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td>O</td>
<td>[14]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>[15]</td>
<td>Tentatively reserved for Packed-SIMD extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>[16]</td>
<td>Quad-precision floating-point extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>[17]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>[18]</td>
<td>Supervisor mode implemented</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>[19]</td>
<td>Tentatively reserved for Transactional Memory extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>[20]</td>
<td>User mode implemented</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td>[21]</td>
<td>Tentatively reserved for Vector extension</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>[22]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>[23]</td>
<td>Non-standard extensions present</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>[24]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>[25]</td>
<td>Reserved</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>Base</td>
<td>[63:62]</td>
<td>The general-purpose register width of the native base integer ISA.</td>
<td>RO</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

### 16.3.3 Machine Exception Delegation

**Mnemonic Name:** medeleg
**IM Requirement:** Required
**Access Mode:** Machine
**CSR Address:** 0x302 (standard read/write)
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM</td>
<td>[0]</td>
<td>IAM indicates whether an Instruction Address Misaligned exception will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

| IAF    | [1] | IAF indicates whether an Instruction Access Fault exception will be delegated to S-mode. | RW   | 0     |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

| II     | [2] | II indicates whether an Illegal Instruction exception will be delegated to S-mode. | RW   | 0     |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

| B      | [3] | B indicates whether an exception triggered by breakpoint will be delegated to S-mode. | RW   | 0     |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

| LAM    | [4] | LAM indicates whether a Load Address Misaligned exception will be delegated to S-mode. | RW   | 0     |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAF</td>
<td>[5]</td>
<td>LAF indicates whether a Load Access Fault exception will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>[6]</td>
<td>SAM indicates whether a Store/AMO Address Misaligned exception will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SAF</td>
<td>[7]</td>
<td>SAF indicates whether a Store/AMO Access Fault exception will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>UEC</td>
<td>[8]</td>
<td>UEC indicates whether an exception triggered by environment call from U-mode will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>[9]</td>
<td>SEC indicates whether an exception triggered by environment call from S-mode will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
### Field Name Bits Description Type Reset

**IPF** [12]  
**IPF** indicates whether an Instruction Page Fault exception will be delegated to S-mode.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

**LPF** [13]  
**LPF** indicates whether a Load Page Fault exception will be delegated to S-mode.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

**SPF** [15]  
**SPF** indicates whether a Store/AMO Page Fault exception will be delegated to S-mode.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not delegate</td>
</tr>
<tr>
<td>1</td>
<td>delegate</td>
</tr>
</tbody>
</table>

When supervisor mode or N extension is not supported, the corresponding bits in medeleg are hard-wired to zero.

### 16.3.4 Machine Interrupt Delegation

**Mnemonic Name:** mideleg  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x303 (standard read/write)
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USI</td>
<td>[0]</td>
<td>USI indicates whether an U-mode software interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SSI</td>
<td>[1]</td>
<td>SSI indicates whether an S-mode software interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>UTI</td>
<td>[4]</td>
<td>UTI indicates whether an U-mode timer interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>[5]</td>
<td>STI indicates whether an S-mode timer interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>UEI</td>
<td>[8]</td>
<td>UEI indicates whether an U-mode external interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SEI</td>
<td>[9]</td>
<td>SEI indicates whether an S-mode external interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
</tbody>
</table>
When supervisor mode or N extension is not supported, the corresponding bits in mideleg are hard-wired to zero.

### 16.3.5 Machine Interrupt Enable

**Mnemonic Name:** mie  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x304 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIE</td>
<td>[0]</td>
<td>U-mode software interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SSIE</td>
<td>[1]</td>
<td>S-mode software interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UTIE</td>
<td>[4]</td>
<td>U-mode timer interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>STIE</td>
<td>[5]</td>
<td>S-mode timer interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Value Meaning

- **Value**
- **Meaning**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMODE</td>
<td>[8]</td>
<td>U-mode external interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SMODE</td>
<td>[9]</td>
<td>S-mode external interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>IMECCI</td>
<td>[16]</td>
<td>Imprecise ECC error local interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>BWEI</td>
<td>[17]</td>
<td>Bus read/write transaction error local interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

The processor may receive bus errors on load/store instructions or cache writebacks.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
When supervisor mode or N extension is not supported, the corresponding bits in mie are hardwired to zero.

Each local interrupt can be configured with a local interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields above show the default bit location.

### 16.3.6 Machine Trap Vector Base Address

**Mnemonic Name:** mtvec  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x305 (standard read/write)

This register determines the base address of the trap vector. The least significant 2 bits are hardwired to zeros. When the configured address width is less than 64, the upper bits are hardwired to zeros. When mmisc_ctl.VEC_PLIC is 0 (PLIC is not in the vector mode), this register indicates the entry points for the trap handler and it may point to any 4-byte aligned location in the memory space.

On the other hand, when mmisc_ctl.VEC_PLIC is 1 (PLIC is in the vector mode), this register will be the base address of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.

- This register should be aligned to $2^{\log_2 N+2}$-byte boundary for PLIC with $N$ interrupt sources. For example, if $N$ is 1023, the minimum alignment requirement is 4096 bytes (4 KiB).
- mtvec[0] is for exceptions and non-external local interrupts.
- mtvec[i] is for external PLIC interrupt source $i$ triggered through the mip.MEIP pending condition.
- mtvec[1024+i] is for external PLIC interrupt source $i$ triggered through
  - the mip.SEIP pending condition when mideleg.SEI == 0 for M/S/U systems.
– the mip.UEIP pending condition when mideleg.UEI == 0 for M/U systems.

• mtvec[2048+i] is for external PLIC interrupt source i triggered through the mip.UEIP pending condition when mideleg.UEI == 0 for M/S/U systems.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE[63:2]</td>
<td>[63:2]</td>
<td>Base address for interrupt and exception handlers. See description above for alignment requirements when PLIC is in the vector mode.</td>
</tr>
</tbody>
</table>

### 16.3.7 Machine Scratch Register

**Mnemonic Name:** mscratch  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x340 (standard read/write)

This is a scratch register for temporary data storage, which is typically used by the M-mode trap handler.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSCRATCH</td>
<td>[63:0]</td>
<td>Scratch register storage.</td>
</tr>
</tbody>
</table>

### 16.3.8 Machine Exception Program Counter

**Mnemonic Name:** mepc  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x341 (standard read/write)

This register is written with the virtual address of the instruction that encountered traps and/or NMIs when these events occurred.
Field Name | Bits   | Description               | Type | Reset |
---         | ---    | --------------------------|------|-------|
EPC        | [63:1] | Exception program counter. | RW   | 0     |

16.3.9 Machine Cause Register

Mnemonic Name: mcause
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x342 (standard read/write)

This register indicates the cause of trap, reset, NMI or the interrupt source ID of a vector interrupt. This register is updated when a trap, reset, NMI or vector interrupt occurs. Please see Section 13 for an overview of how interrupts and exceptions are handled by AX45MP-1C. When multiple events may cause a trap to be taken with the same mcause value, the value of mdcause records the exact event that causes the trap.

Exceptions can be precise or imprecise. Only precise exceptions are triggered as the standard RISC-V exceptions with the mcause.INTERRUPT bit clear. Imprecise exceptions are triggered as local interrupts, with the mcause.INTERRUPT bit set.

Field Name | Bits | Description | Type | Reset |
---         | ---  | ----------- |------|-------|
EXCEPTION_CODE | [11:0] | Exception code | RW   | 0     |
INTERRUPT   | [63]  | Interrupt   | RW   | 0     |

The following tables show the possible values of mcause:

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>User software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>User timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Machine timer interrupt</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 50: (continued)

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>User external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Supervisor external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Machine external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>Imprecise ECC error interrupt (slave port accesses, D-Cache evictions, and nonblocking load/stores) (M-mode)</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>Bus read/write transaction error interrupt (M-mode)</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>Performance monitor overflow interrupt (M-mode)</td>
</tr>
<tr>
<td>1</td>
<td>256+16</td>
<td>Imprecise ECC error interrupt (slave port accesses, D-Cache evictions, and nonblocking load/stores) (S-mode)</td>
</tr>
<tr>
<td>1</td>
<td>256+17</td>
<td>Bus write transaction error interrupt (S-mode)</td>
</tr>
<tr>
<td>1</td>
<td>256+18</td>
<td>Performance monitor overflow interrupt (S-mode)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>Environment call from S-mode</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>Environment call from M-mode</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>Instruction page fault</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>Load page fault</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>Store/AMO page fault</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>Stack overflow exception</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>Stack underflow exception</td>
</tr>
</tbody>
</table>

Table 51: Possible Values of mcause After Reset

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Initial value when the processor comes out of reset</td>
</tr>
</tbody>
</table>
Table 52: Possible Values of mcause After NMI

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x001</td>
<td>NMI triggered</td>
</tr>
</tbody>
</table>

Table 53: Possible Values of mcause After Vector Interrupt

<table>
<thead>
<tr>
<th>mcause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt source ID when a vector interrupt occurs</td>
</tr>
</tbody>
</table>

16.3.10 Machine Trap Value

**Mnemonic Name:** mtval  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x343 (standard read/write)

This register is updated when a trap is taken to M-mode. The updated value is dependent on the cause of traps:

- For Hardware Breakpoint exceptions, Address Misaligned exceptions, Page Fault exceptions, or Access Fault exceptions, it is the effective faulting addresses.

- For illegal instruction exceptions, the updated value is the faulting instruction. If the length of the instruction is less than XLEN bits long, the upper bits of mtval are cleared to zero. For the EXEC.IT instructions triggering illegal instruction exceptions, the faulting instruction is the translated instruction. Please note that if a EXEC.IT instruction is translated to a 16-bit instruction, the translated instruction is considered an illegal instruction even if it is normally a valid one.

- For other exceptions, mtval is set to zero.

For instruction-fetch access faults, this register will be updated with the address pointing to the portion of the instruction that caused the fault, while the mepc register will be updated with the address pointing to the beginning of the instruction.

When the configured address width is less than 64, the upper bits of mtval are hardwired to zeros.
Field Name | Bits | Description | Type | Reset
---|---|---|---|---
MTVAL | [63:0] | Exception-specific information for software trap handling. | RW | 0

### 16.3.11 Machine Interrupt Pending

**Mnemonic Name:** mip  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x344 (standard read/write)

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>PMOVI</td>
</tr>
<tr>
<td>19</td>
<td>BWEI</td>
</tr>
<tr>
<td>18</td>
<td>IMECCI</td>
</tr>
<tr>
<td>11</td>
<td>MEIP</td>
</tr>
<tr>
<td>10</td>
<td>SEIP</td>
</tr>
<tr>
<td>9</td>
<td>UEIP</td>
</tr>
<tr>
<td>8</td>
<td>MTIP</td>
</tr>
<tr>
<td>7</td>
<td>STIP</td>
</tr>
<tr>
<td>6</td>
<td>UTIP</td>
</tr>
<tr>
<td>3</td>
<td>MSIP</td>
</tr>
<tr>
<td>1</td>
<td>SSIP</td>
</tr>
<tr>
<td>0</td>
<td>USIP</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
</table>
| USIP | [0] | U-mode software interrupt pending bit. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not pending</td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
</tr>
</tbody>
</table>

| SSIP | [1] | S-mode software interrupt pending bit. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not pending</td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not pending</td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
</tr>
</tbody>
</table>

| UTIP | [4] | U-mode timer interrupt pending bit. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not pending</td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>STIP</td>
<td>[5]</td>
<td>S-mode timer interrupt pending bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UEIP</td>
<td>[8]</td>
<td>U-mode external interrupt pending bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEIP</td>
<td>[9]</td>
<td>S-mode external interrupt pending bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMECCI</td>
<td>[16]</td>
<td>Imprecise ECC error local interrupt pending bit. The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not pending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWEI</td>
<td>[17]</td>
<td>Bus read/write transaction error local interrupt pending bit. The processor may receive bus errors on load/store instructions or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not pending</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Pending</td>
<td></td>
</tr>
<tr>
<td>PMOVI</td>
<td>[18]</td>
<td>Performance monitor overflow local interrupt pending bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not pending</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Pending</td>
<td></td>
</tr>
</tbody>
</table>

When supervisor mode or N extension is not supported, the corresponding bits in `mip` are hardwired to zero.

Each local interrupt can be configured with a local interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields above show the default bit location.

16.3.12 Machine Extended Status

Mnemonic Name: mxstatus
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7c4 (non-standard read/write)
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFT_EN</td>
<td>[0]</td>
<td>Enable performance throttling. When throttling is enabled, the processor executes instructions at the performance level specified in mpft_ctl.T_LEVEL. On entering a trap: • PPFT_EN ← PFT_EN; • PFT_EN ← mpft_ctl.FAST_INT ? 0 : PFT_EN; On executing an MRET instruction: • PFT_EN ← PPFT_EN; This field is hardwired to 0 if the PowerBrake feature is not supported.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>PPFT_EN</td>
<td>[1]</td>
<td>For saving previous PFT_EN state on entering a trap. This field is hardwired to 0 if the PowerBrake feature is not supported.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>DME</td>
<td>[4]</td>
<td>Data Machine Error flag. It indicates an exception occurred at the data cache or data local memory (DLM). Load/store accesses will bypass D-Cache when this bit is set. The exception handler should clear this bit after the machine error has been dealt with.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>PDME</td>
<td>[5]</td>
<td>For saving previous DME state on entering a trap. This field is hardwired to 0 if data cache and data local memory are not supported.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

### 16.3.13 Machine Detailed Trap Cause

**Mnemonic Name:** mdcause  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x7c9 (non-standard read/write)

For precise exceptions:

```
+---+---+---+---+---+  
|63|32|23|14|0  
+---+---+---+---+---+  
| 0 |    |    |    | MDCAUSE  
+---+---+---+---+---+  
```

For imprecise exceptions (local interrupts):
When multiple events cause traps to be taken with the same mcause value, this register helps to further disambiguate them. Some events could cause either precise exceptions or imprecise exceptions (local interrupts) depending on when they are detected, so they can appear in multiple tables below.

Imprecise exceptions are triggered as local interrupts, so the tables below for mcause == Local Interrupt n summarizes imprecise exceptions delivered as local interrupt n. The mcause == Local Interrupt n notation stands for (INTERRUPT, EXCEPTION_CODE) fields of mcause is (1, n).

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDCAUSE</td>
<td>[2:0]</td>
<td>This register further disambiguates causes of traps recorded in the mcause register. See the list below for details.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>PM</td>
<td>[6:5]</td>
<td>When mcause is imprecise exception (in the form of an interrupt), the PM field records the privileged mode of the instruction that caused the imprecise exception. The PM field encoding is defined as follows:</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>User mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Supervisor mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Machine mode</td>
<td></td>
</tr>
</tbody>
</table>

The value of MDCAUSE for precise exception:

- When mcause == 1 (Instruction access fault):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP instruction access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>PMA empty hole access</td>
</tr>
</tbody>
</table>

- When mcause == 2 (Illegal instruction):
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The actual faulting instruction is stored in the mtval CSR.</td>
</tr>
<tr>
<td>1</td>
<td>FP disabled exception</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- When `mcause` == 5 (Load access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP load access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>5</td>
<td>PMA empty hole access</td>
</tr>
<tr>
<td>6</td>
<td>PMA attribute inconsistency</td>
</tr>
<tr>
<td>7</td>
<td>PMA NAMO exception</td>
</tr>
</tbody>
</table>

- When `mcause` == 7 (Store access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP store access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>5</td>
<td>PMA empty hole access</td>
</tr>
<tr>
<td>6</td>
<td>PMA attribute inconsistency</td>
</tr>
<tr>
<td>7</td>
<td>PMA NAMO exception</td>
</tr>
</tbody>
</table>

The value of `MDCAUSE` for imprecise exception:

- When `mcause` == `Local Interrupt 16` or `Local Interrupt 272 (16 + 256)` (ECC error local interrupt)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>LM slave port ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>Imprecise store ECC/Parity error</td>
</tr>
<tr>
<td>3</td>
<td>Imprecise load ECC/Parity error</td>
</tr>
</tbody>
</table>

- When `mcause == Local Interrupt 17` or `Local Interrupt 273 (17 + 256)` (Bus read/write transaction error local interrupt)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Bus read error</td>
</tr>
<tr>
<td>2</td>
<td>Bus write error</td>
</tr>
<tr>
<td>3</td>
<td>PMP error caused by load instructions</td>
</tr>
<tr>
<td>4</td>
<td>PMP error caused by store instructions</td>
</tr>
<tr>
<td>5</td>
<td>PMA error caused by load instructions”</td>
</tr>
<tr>
<td>6</td>
<td>PMA error caused by store instructions</td>
</tr>
</tbody>
</table>

- For `PMOVI`, `MDCAUSE` will be written 0. For other exceptions and interrupts, this register will not be updated.

### 16.3.13.1 Detailed Exception Priority

Within Instruction/Load/Store access fault exceptions, the priority of a PMP exception is higher than the priority of a PMA exception, when both types of exceptions happen on the same instruction.

### 16.3.14 Machine Supervisor Local Interrupt Delegation

**Mnemonic Name:** `mslideleg`  
**IM Requirement:** `misa[18] = = 1`  
**Access Mode:** Machine  
**CSR Address:** `0x7D5` (non-standard read/write)
This register controls the delegation of supervisor local interrupts. If a supervisor local interrupt is not delegated, the supervisor local interrupt will be taken in M-mode. If a supervisor local interrupt is delegated, the supervisor local interrupt will be taken in S-mode.

The privileged mode of IMECCI and BWEI are determined by the current privileged mode. For M/S/U configuration, if the current privileged mode is User or Supervisor, the local interrupt generated is a supervisor local interrupt. For M/U configuration, if the current privileged mode is User, the local interrupt generated is a machine local interrupt. The privileged mode of the above PMOVI interrupt is determined by the counter state of mcountermask_m.

Each local interrupt can be configured with a local interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields below show the default bit location.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMECCI</td>
<td>[16]</td>
<td>Delegate S-mode imprecise ECC error local interrupt to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BWEI</td>
<td>[17]</td>
<td>Delegate S-mode bus read/write transaction error local interrupt to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMOVI</td>
<td>[18]</td>
<td>Delegate S-mode performance monitor overflow local interrupt to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Delegate to S-mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
16.4 Counter Related CSRs

16.4.1 Machine Cycle Counter

Mnemonic Name: mcycle
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb00 (standard read/write)

The mcycle CSR counts the number of cycles that the hart has executed since some arbitrary time in the past. The mcycle register has 64-bit precision.
16.4.2 Machine Instruction-Retired Counter

Mnemonic Name: minstret
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb02 (standard read/write)

The minstret CSR counts the number of instructions that the hart has retired since some arbitrary time in the past. The minstret register has 64-bit precision.

16.4.3 Machine Performance Monitoring Counter

Mnemonic Name: mhpmccounter3–mhpmccounter6
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb03 to 0xb06 (standard read/write)

The mhpmccounter3–mhpmccounter6 CSRs count the number of events selected by mhpmevent3–mhpmevent6.

16.4.4 Machine Counter-Inhibit

Mnemonic Name: mcountinhibit
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x320 (non-standard read/write)

The counter-inhibit register controls which counters should not be incremented. When the CY, IR, or HPMn bit is set, the corresponding counter will not be incremented on the event.

16.4.5 Machine Performance Monitoring Event Selector

Mnemonic Name: mhpmevent3–mhpmevent6
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x323 to 0x326 (standard read/write)
The event selectors are defined in Table 54. Micro-architectural events are mostly speculative in nature. The counted events include events caused by speculative actions, unless they are defined to be non-speculative in the comment section. In particular, retired instruction counts are non-speculative.

### Table 54: Event Selectors

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEL</th>
<th>Event Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Cycle count</td>
<td>Number of elapsed processor clock cycles.</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Retired instruction count</td>
<td>Number of retired instructions.</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Integer load instruction count</td>
<td>Number of retired load instructions (including LR).</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Integer store instruction count</td>
<td>Number of retired store instructions (including SC).</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Atomic instruction count</td>
<td>Number of retired atomic instructions (not including LR and SC).</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>System instruction count</td>
<td>Number of retired SYSTEM instructions (instructions with major opcode equal to 0b1110011).</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Integer computational instruction count</td>
<td>Number of retired integer computational instructions.</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Conditional branch instruction count</td>
<td>Number of retired conditional branch instructions.</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>Taken conditional branch instruction count</td>
<td>Number of retired conditional branch instructions that are taken.</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>JAL instruction count</td>
<td>Number of retired JAL instructions.</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>JALR instruction count</td>
<td>Number of retired JALR instructions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This event selector also counts the events monitored by the return instruction count event selector defined in the next row.</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>Return instruction count</td>
<td>Number of retired return instructions. Return instructions are JALR instructions with zero immediate offset and the following operands: • (rd != x1/x5) and (rs1 == x1/x5) • rd == x1 and rs1 == x5 • rd == x5 and rs1 == x1</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>Control transfer instruction count</td>
<td>Number of retired unconditional jumps (JAL and JALR) and conditional branch instructions.</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>EXEC.IT instruction count</td>
<td>Number of retired EXEC.IT instructions.</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>Integer multiplication instruction count</td>
<td>Number of retired integer multiplication instructions.</td>
</tr>
</tbody>
</table>
Table 54: (continued)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEL</th>
<th>Event Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>Integer division instruction count</td>
<td>Number of retired integer division/remainder instructions.</td>
</tr>
<tr>
<td>0</td>
<td>17</td>
<td>Floating-point load instruction count</td>
<td>Number of retired floating-point load instructions.</td>
</tr>
<tr>
<td>0</td>
<td>18</td>
<td>Floating-point store instruction count</td>
<td>Number of retired floating-point store instructions.</td>
</tr>
<tr>
<td>0</td>
<td>19</td>
<td>Floating-point addition instruction count</td>
<td>Number of retired floating-point addition/subtraction instructions.</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>Floating-point multiplication instruction count</td>
<td>Number of retired floating-point multiplication instructions.</td>
</tr>
<tr>
<td>0</td>
<td>21</td>
<td>Floating-point fused multiply-add instruction count</td>
<td>Number of retired floating-point fused multiply-add/subtraction instructions (FMADD, FMSUB, FNMSUB, FNMADD).</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>Floating-point division or square-root instruction count</td>
<td>Number of retired floating-point division/square-root instructions.</td>
</tr>
<tr>
<td>0</td>
<td>23</td>
<td>Other floating-point instruction count</td>
<td>Number of retired floating-point instructions not counted by the previous floating-point instruction event selectors.</td>
</tr>
<tr>
<td>0</td>
<td>24</td>
<td>Integer multiplication and add/sub instruction count</td>
<td>Number of retired integer multiplication and add/sub instructions.</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>Retired operation count</td>
<td>Number of retired operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• a floating-point multiply-add instruction is counted as 2 operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• other instructions are counted as 1 operation.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>ILM access</td>
<td>Number of ILM transfers, including speculative instruction fetch, load/store accesses, ECC repair and slave port accesses.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>DLM access</td>
<td>Number of DLM transfers, including speculative load/store accesses, ECC repair and slave port accesses.</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>I-Cache access</td>
<td>Number of completed I-Cache fetch access.</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>I-Cache miss</td>
<td>Number of I-Cache fetch miss.</td>
</tr>
</tbody>
</table>

Continued on next page…
Table 54: (continued)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEL</th>
<th>Event Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>D-Cache access*</td>
<td>Number of completed D-Cache load-and-store access. Misaligned load/store accesses might increase this counter by either one or two, depending on access sizes and alignments. Only misaligned accesses crossing two cache lines are guaranteed to result in increment of two.</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>D-Cache miss*</td>
<td>The event counts the number of D-Cache load-and-store miss. Misaligned load/store accesses might increase this counter by either zero, one or two, depending on access sizes, alignments and whether the accessed lines are in D-Cache.</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>D-Cache load access*</td>
<td>Number of completed D-Cache load access. See the D-Cache access count event selector for the handling of misaligned load accesses.</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>D-Cache load miss*</td>
<td>Number of D-Cache load miss. See the D-Cache miss count event selector for the handling of misaligned load accesses.</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>D-Cache store access*</td>
<td>Number of completed D-Cache store access. See the D-Cache access count event selector for the handling of misaligned load accesses.</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>D-Cache store miss*</td>
<td>Number of D-Cache store miss. See the D-Cache miss count event selector for the handling of misaligned load accesses.</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>D-Cache writeback*</td>
<td>Number of D-Cache writeback.</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Cycles waiting for I-Cache fill data*</td>
<td>Number of cycles waiting for the return of the critical word of I-Cache misses from the system bus. This event selector does not monitor accesses to I/O regions or accesses to cacheable regions when I-Cache is turned off.</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 54: (continued)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEL</th>
<th>Event Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>Cycles waiting for D-Cache fill data*</td>
<td>Number of cycles waiting for the return of the critical word of D-Cache misses from the system bus. This event selector does not monitor accesses to I/O regions or accesses to cacheable regions when D-Cache is turned off. Additionally, non-blocking loads do not increment this counter since they do not cause pipeline stalls under D-Cache misses.</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>Uncached fetch data access from bus*</td>
<td>Number of accesses of uncached instruction data returning from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when I-Cache is not configured or off.</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>Uncached load data access from bus*</td>
<td>Number of accesses of uncached load data returning from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when D-Cache is not configured or off.</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>Cycles waiting for uncached fetch data from bus*</td>
<td>Number of cycles waiting for the instruction data to return from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when I-Cache is not configured or off.</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>Cycles waiting for uncached load data from bus*</td>
<td>Number of cycles waiting for the load data to return from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when D-Cache is not configured or off.</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>Main ITLB access</td>
<td>Number of address translation requests performed by the shared TLB for instruction fetches</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>Main ITLB miss</td>
<td>Number of address translation requests from instruction fetch that miss Shared TLB and invoke the hardware page table walker.</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>Main DTLB access</td>
<td>Number of address translation requests performed by the shared TLB for load/store accesses</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 54: (continued)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEL</th>
<th>Event Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>Main DTLB miss</td>
<td>Number of address translation requests from load/store accesses that miss Shared TLB and invoke the hardware page table walker.</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>Cycles waiting for Main ITLB fill data</td>
<td>Number of instruction fetch stall cycles attributable to TLB misses.</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>Pipeline stall cycles caused by Main DTLB miss</td>
<td>Number of pipeline stall cycles attributable to address translation for load/store accesses.</td>
</tr>
<tr>
<td>1</td>
<td>23</td>
<td>Hardware prefetch bus access</td>
<td>This event counts the bus accesses generated by the hardware data prefetcher.</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Misprediction of conditional branches (direction)</td>
<td>Number of misprediction of committed conditional branches.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Misprediction of taken conditional branches (direction)</td>
<td>Number of misprediction of committed taken conditional branches.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Misprediction of targets of Return instructions</td>
<td>Number of misprediction of committed Return instruction.</td>
</tr>
</tbody>
</table>

Note

- Interrupts are expected to be disabled when monitoring D-Cache related events and cycles waiting related events.

16.4.6 Machine Counter Enable

Mnemonic Name: mcounteren
IM Requirement: Required if User mode is implemented
Access Mode: Machine
CSR Address: 0x306 (standard read/write)

```
63 7 6 5 4 3 2 1 0
    0 HPM6 HPM5 HPM4 HPM3 IR TM CY
```

The machine counter-enable register controls the availability of the hardware performance monitoring counters to the next-lowest privileged mode. The default value of this register is 0.
When CY, TM, IR, HPM3, HPM4, HPM5, or HPM6 in the mcounteren register is 0, attempts to read
the cycle, time, instret, hpmcounter3, hpmcounter4, hpmcounter5, or hpmcounter6 registers
while executing in U-mode (M/U configuration) or S-mode (M/S/U configuration) will cause an
illegal instruction exception. When one of these bits is set, accessing to the corresponding register is
permitted in the next implemented privilege mode.

16.4.7 Machine Counter Write Enable

Mnemonic Name: mcounterwen
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7CE (non-standard read/write)

The machine counter write enable register controls the permission of writing the hardware perfor-
mance monitoring counters in the next-lowest privileged mode and M-mode itself. The default value
of this register is 0.

When CY, IR, HPM3, HPM4, HPM5, or HPM6 in the mcounterwen register is 0, attempts to write the
cycle, time, instret, hpmcounter3, hpmcounter4, hpmcounter5, or hpmcounter6 registers
while executing in U-mode or M-mode (M/U configuration) or S-mode (M/S/U configuration)
will cause an illegal instruction exception. When one of these bits is set, writing to the corresponding
register is permitted in M-mode and the next implemented privilege mode.

Note
The register does not affect permissions of M-Mode counters, i.e. mcycle, mtime, minstret, mhpmco-
tunter3, mhpmcounter4, and mhpmcounter5.

16.4.8 Machine Counter Interrupt Enable

Mnemonic Name: mcounterinten
IM Requirement: mmsc_cfg.PMNDS == 1
Access Mode: Machine
CSR Address: 0x7CF (non-standard read/write)

The machine counter interrupt enable register controls whether a counter overflow interrupt is gen-
erated or not. The default value of this register is 0.
When CY, IR, HPM3, HPM4, HPM5, or HPM6 in the mcounterinten register is 0, no overflow interrupt is generated for the corresponding counter. When one of these bits is set, an interrupt will be generated when the corresponding counter overflows (the counter value wraps around back to 0).

16.4.9 Machine Counter Mask for Machine Mode

Mnemonic Name: mcountermask_m
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7D1 (non-standard read/write)

<table>
<thead>
<tr>
<th>63</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HPM6</td>
<td>HPM5</td>
<td>HPM4</td>
<td>HPM3</td>
<td>IR</td>
<td>0</td>
<td>CY</td>
<td></td>
</tr>
</tbody>
</table>

The machine counter mask for M-mode register controls the performance counter behavior in M-mode. The default value of this register is 0.

When CY, IR, HPM3, HPM4, HPM5, or HPM6 in the mcountermask_m register is set, the specific counter will not be incremented in M-mode.

The setting in this register also controls the privileged mode of the overflow local interrupt when the corresponding counter overflows for the M/S/U configuration: For any bit in this register, overflow of the corresponding counter will trigger an M-mode interrupt if the bit is zero and an S-mode interrupt if the bit is one.

On the other hand, a counter overflow will always generate an M-mode interrupt for the M/U configuration, regardless of the settings in this register.

16.4.10 Machine Counter Mask for Supervisor Mode

Mnemonic Name: mcountermask_s
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[18] == 1
Access Mode: Machine
CSR Address: 0x7D2 (non-standard read/write)

<table>
<thead>
<tr>
<th>63</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HPM6</td>
<td>HPM5</td>
<td>HPM4</td>
<td>HPM3</td>
<td>IR</td>
<td>0</td>
<td>CY</td>
<td></td>
</tr>
</tbody>
</table>

The machine counter mask for S-mode register controls the performance counter behavior in S-mode. The default value of this register is 0.
16.4.11 Machine Counter Mask for User Mode

Mnemonic Name: mcountermask_u
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7D3 (non-standard read/write)

The machine counter mask for U-mode register controls the performance counter behavior in U-mode. The default value of this register is 0.

16.4.12 Machine Counter Overflow Status

Mnemonic Name: mcounterovf
IM Requirement: mmsc_cfg.PMNDS == 1
Access Mode: Machine
CSR Address: 0x7D4 (non-standard read/write)

The machine counter overflow status register records the overflow status of performance counters. When a bit is set, it indicates that an overflow has happened to the corresponding counter. Writing 0 will deassert the corresponding bit and writing 1 will have no effect.
16.5 Configuration Control & Status Registers

16.5.1 Instruction Cache/Memory Configuration Register

Mnemonic Name: micm_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc0 (non-standard read only)

This register provides information about configurations of instruction cache and instruction memory.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISET</td>
<td>[2:0]</td>
<td>I-Cache sets (# of cache lines per way):</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>

When micm_cfg.SETH==0:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
</tr>
<tr>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>5</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>4096</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

When micm_cfg.SETH==1:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3~7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- When instruction cache is not configured, this field should be ignored.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWAY</td>
<td>[5:3]</td>
<td>Associativity of I-Cache</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Direct-mapped</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>6-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>8-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When instruction cache is not configured, this field should be ignored.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISZ</th>
<th>[8:6]</th>
<th>I-Cache block (line) size</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No I-Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>8 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>16 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>32 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>64 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>128 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6,7</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When instruction cache is not configured, this field should be ignored.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ILCK</th>
<th>[9]</th>
<th>I-Cache locking support</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No locking support</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>With locking support</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When instruction cache is not configured, this field should be ignored.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_ECC</td>
<td>[11:10]</td>
<td>I-Cache soft-error protection scheme</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>No parity/ECC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>ECC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• When instruction cache is not configured, this field should be ignored.

<table>
<thead>
<tr>
<th>ILMB</th>
<th>[14:12]</th>
<th>Number of ILM base registers present</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>No ILM base register present</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>One ILM base register present</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2-7</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• When ILM is not configured, this field should be ignored.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILMSZ</td>
<td>[19:15]</td>
<td>ILM Size</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0 Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>8 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>16 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>32 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>64 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>128 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>256 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>512 KiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>1 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>2 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>4 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>8 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>16 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-31</td>
<td></td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - When ILM is not configured, this field should be ignored.

<table>
<thead>
<tr>
<th>ILM_ECC</th>
<th>[22:21]</th>
<th>ILM soft-error protection scheme</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>No parity/ECC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>ECC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - When ILM is not configured, this field should be ignored.

<table>
<thead>
<tr>
<th>SETH</th>
<th>[24]</th>
<th>This bit extends the ISET field.</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
</table>

  - When instruction cache is not configured, this field should be ignored.

<table>
<thead>
<tr>
<th>IC_REPL</th>
<th>[26:25]</th>
<th>This field is hardwired to zeros.</th>
<th>RO</th>
<th>0</th>
</tr>
</thead>
</table>
16.5.2 Data Cache/Memory Configuration Register

Mnemonic Name: mdcm_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc1 (non-standard read only)

This register provides information about the configurations of data cache and data local memory.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSET</td>
<td>[2:0]</td>
<td>D-Cache sets (# of cache lines per way):</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
</tr>
<tr>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>5</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>4096</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- When data cache is not configured, this field should be ignored.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWay</td>
<td>[5:3]</td>
<td>Associativity of D-Cache</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Direct-mapped</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>6-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>8-way</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>• When data cache is not configured, this field should be ignored.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dsz</td>
<td>[8:6]</td>
<td>D-Cache block (line) size</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No D-Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>8 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>16 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>32 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>64 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>128 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6,7</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>• When data cache is not configured, this field should be ignored.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dlck</td>
<td>[9]</td>
<td>D-Cache locking support</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No locking support</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>With locking support</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>• When data cache is not configured, this field should be ignored.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Name</td>
<td>Bits</td>
<td>Description</td>
<td>Type</td>
<td>Reset</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------------------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No parity/ECC support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Has parity support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Has ECC support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLMB</td>
<td>[14:12]</td>
<td>Number of DLM base registers present</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No DLM base register present</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>One DLM base register present</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-7</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- When data cache is not configured, this field should be ignored.
- When DLM is not configured, this field should be ignored.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLMSZ</td>
<td>[19:15]</td>
<td>DLM Size</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0 Byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>8 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>16 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>32 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>64 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>128 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>256 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>512 KiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>1 MiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>2 MiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>4 MiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>8 MiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>16 MiB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16-31</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLM_ECC</td>
<td>[22:21]</td>
<td>DLM soft-error protection scheme</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No parity/ECC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Parity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>ECC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETH</td>
<td>[24]</td>
<td>This bit extends the DSET field</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC_REPL</td>
<td>[26:25]</td>
<td>This field is hardwired to zeros.</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>
16.5.3 Misc. Configuration Register

Mnemonic Name: mmsc_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc2 (non-standard read only)

This register provides information regarding miscellaneous processor configurations.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC</td>
<td>[0]</td>
<td>Indicates whether the parity/ECC soft-error protection is implemented or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The specific parity/ECC scheme used for each protected RAM is specified by the control bits in the following list.

- micm_cfg.IC_ECC
- micm_cfg.ILM_ECC
- mdcm_cfg.DC_ECC
- mdcm_cfg.DLM_ECC
- mmsc_cfg.TLB_ECC

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB_ECC</td>
<td>[2:1]</td>
<td>TLB parity/ECC support configuration.</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>No parity/ECC support.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Has parity support.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Has ECC support.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECD</td>
<td>[3]</td>
<td>Indicates whether the Andes CoDense Extension is implemented or not.</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFT</td>
<td>[4]</td>
<td>Indicates whether the Andes PowerBrake (Performance Throttling) power/performance scaling extension is implemented or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HSP</td>
<td>[5]</td>
<td>Indicates whether the Andes StackSafe hardware stack protection extension is implemented or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPLIC</td>
<td>[12]</td>
<td>Indicates whether the Andes Vectored PLIC Extension is implemented or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Not implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Implemented.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV5PE</td>
<td>[13]</td>
<td>Indicates whether AndeStar V5 Performance Extension is implemented or not. AX45MP-1C always implements AndeStar V5 Performance Extension.</td>
<td>RO</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMSLVP</td>
<td>[14]</td>
<td>Indicates if local memory slave port is present or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Local memory slave port is not present.</td>
</tr>
<tr>
<td>1</td>
<td>Local memory slave port is implemented.</td>
</tr>
</tbody>
</table>

Note that atomicity of atomic instructions accessing local memory address space is not guaranteed if external masters modify the same data through the local memory slave port.

<table>
<thead>
<tr>
<th>PMNDS</th>
<th>[15]</th>
<th>Indicates if Andes performance monitor extension is present or not. This extension should be present when Performance Monitors are configured.</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Andes-enhanced performance monitoring feature is not supported.</td>
</tr>
<tr>
<td>1</td>
<td>Andes-enhanced performance monitoring feature is supported.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CCTLCSR</th>
<th>[16]</th>
<th>Indicates the presence of CSRs for CCTL operations.</th>
<th>RO</th>
<th>IM</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Feature of CSRs for CCTL operations is not supported.</td>
</tr>
<tr>
<td>1</td>
<td>Feature of CSRs for CCTL operations is supported.</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFHW</td>
<td>[17]</td>
<td>Indicates the support of FLHW and FSHW instructions.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>FLHW and FSHW instructions are not supported.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>FLHW and FSHW instructions are supported.</td>
<td></td>
</tr>
<tr>
<td>VCCTL</td>
<td>[19:18]</td>
<td>Indicates the version number of CCTL command operation scheme supported by an implementation.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Instruction cache and data cache are not configured.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Instruction cache or data cache is configured.</td>
<td></td>
</tr>
<tr>
<td>EDSP</td>
<td>[29]</td>
<td>Indicates if the DSP extension is supported or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>The DSP extension is not supported.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>The DSP extension is supported.</td>
<td></td>
</tr>
<tr>
<td>PPMA</td>
<td>[30]</td>
<td>Indicates if programmable PMA setup with PMA region CSRs is supported or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Programmable PMA setup is not supported.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Programmable PMA setup is supported.</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF16CVT</td>
<td>[32]</td>
<td>Indicates if the BFLOAT16 conversion extension is supported or not.</td>
<td>RO</td>
<td>IM</td>
<td>0</td>
<td>The BFLOAT16 conversion extension is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>The BFLOAT16 conversion extension is supported.</td>
</tr>
<tr>
<td>ZFH</td>
<td>[33]</td>
<td>Indicates if the FP16 half-precision floating-point extension (Zfh) is supported or not.</td>
<td>RO</td>
<td>IM</td>
<td>0</td>
<td>The FP16 extension is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>The FP16 extension is supported.</td>
</tr>
<tr>
<td>FINV</td>
<td>[37]</td>
<td>Indicates if scalar FPU is implemented in VPU.</td>
<td>RO</td>
<td>IM</td>
<td>0</td>
<td>Scalar FPU is not implemented in VPU.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Scalar FPU is implemented in VPU.</td>
</tr>
<tr>
<td>L2CMP_CFG</td>
<td>[45]</td>
<td>Indicates whether cluster configuration info is implemented or not.</td>
<td>RO</td>
<td>IM</td>
<td>0</td>
<td>L2C, IOCP, CORE_PCLUS is not implemented</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>L2C, IOCP, CORE_PCLUS is implemented</td>
</tr>
<tr>
<td>L2C</td>
<td>[46]</td>
<td>Indicates L2-Cache is present or not.</td>
<td>RO</td>
<td>IM</td>
<td>0</td>
<td>L2-Cache is not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>L2-Cache is present</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCP</td>
<td>[47]</td>
<td>Indicates IO Coherence Port is present or not.</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>IOCP is not present</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>IOCP is present</td>
<td></td>
</tr>
<tr>
<td>CORE_PCLUS</td>
<td>[51:48]</td>
<td>Indicates the number of cores in a cluster</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1 core</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2 cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4 cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>8 cores</td>
<td></td>
</tr>
</tbody>
</table>

16.5.4 L2-Cache Control Base Register

**Mnemonic Name:** ml2c_ctl_base  
**IM Requirement:** mmsc_cfg.L2C  
**Access Mode:** Machine  
**CSR Address:** 0xfcf (non-standard read only)

This register indicates the base address of the memory-mapped L2-Cache control registers.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2C_CTL_BASE</td>
<td>[63:0]</td>
<td>Indicates L2C Register Base</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>
16.6 Trigger Registers

16.6.1 Trigger Select

Mnemonic Name: tselect  
IM Requirement: DEBUG_SUPPORT  
Access Mode: Debug and Machine  
CSR Address: 0x7a0 (standard read/write)

This register determines which trigger is accessible through other trigger registers. The setting of accessible triggers must start at 0, and be contiguous. Writes of values greater than or equal to the number of supported triggers might result in a different value in this register than what was written. Debuggers should read back the value to confirm that what they wrote was a valid index.

Since triggers can be used by both Debug mode and Machine mode, the debugger must restore this register after the modification.

16.6.2 Trigger Data 1

Mnemonic Name: tdata1  
IM Requirement: DEBUG_SUPPORT  
Access Mode: Debug and Machine  
CSR Address: 0x7a1 (standard read/write)

This register provides access to the tdata1 register of the currently selected trigger registers selected by the tselect register.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>[58:0]</td>
<td>Trigger-specific data</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMODE</td>
<td>[59]</td>
<td>Setting this field to indicate the trigger is used by Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td>
</tr>
<tr>
<td>1</td>
<td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td>
</tr>
</tbody>
</table>

| TYPE   | [63:60] | Indicates the trigger type. | RW | 2 |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected trigger is invalid.</td>
</tr>
<tr>
<td>2</td>
<td>The selected trigger is an address/data match trigger.</td>
</tr>
<tr>
<td>3</td>
<td>The selected trigger is an instruction count trigger.</td>
</tr>
<tr>
<td>4</td>
<td>The selected trigger is an interrupt trigger.</td>
</tr>
<tr>
<td>5</td>
<td>The selected trigger is an exception trigger.</td>
</tr>
</tbody>
</table>
16.6.3 Trigger Data 2

Mnemonic Name: tdata2
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a2 (standard read/write)

This register provides accesses to the tdata2 register of the currently selected trigger registers selected by the tselect register, and it holds trigger-specific data.
16.6.4 Trigger Data 3

Mnemonic Name: tdata3
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a3 (standard read/write)

This register provides access to the tdata3 register of the currently selected trigger registers selected by the tselect register, and it holds trigger-specific data.
16.6.5 Trigger Info

Mnemonic Name: tinfo
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a4 (standard read/write)

This register provides accesses to the \texttt{tinfo} register of the currently selected trigger registers selected by the \texttt{tselect} register, and it indicates the supported trigger types of the currently selected trigger.
### Field Name | Bits | Description | Type | Reset
---|---|---|---|---
INFO | [15:0] | One bit for each possible type in `tdata1`. Bit $N$ corresponds to type $N$. If the bit is set, then that type is supported by the currently selected trigger. If the currently selected trigger does not exist, this field contains 1. | RO | IM

<table>
<thead>
<tr>
<th>Bit $N$</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When this bit is set, there is no trigger at this <code>tselect</code>.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved and hardwired to 0.</td>
</tr>
<tr>
<td>2</td>
<td>When this bit is set, the selected trigger supports type of address/data match trigger.</td>
</tr>
<tr>
<td>3</td>
<td>When this bit is set, the selected trigger supports type of instruction count trigger.</td>
</tr>
<tr>
<td>4</td>
<td>When this bit is set, the selected trigger supports type of interrupt trigger.</td>
</tr>
<tr>
<td>5</td>
<td>When this bit is set, the selected trigger supports type of exception trigger.</td>
</tr>
<tr>
<td>15</td>
<td>When this bit is set, the selected trigger exists (so enumeration shouldn't terminate), but is not currently available.</td>
</tr>
</tbody>
</table>

Others | Reserved for future use. |

The detailed correlation between trigger types and Number of Triggers are as follows:

- **INFO[2] = 1**, all trigger types are supported.
- **INFO[3] = 1**, trigger 0 or 1 (`tselect = 0` or 1) is supported.
- **INFO[4] = 1**, trigger 0 (`tselect = 0`) or trigger 4 (`tselect = 4`) is supported when Number of Triggers is 8.
- **INFO[5] = 1**,
  - trigger 1 (`tselect = 1`) is supported when Number of Triggers is 2,
– trigger 3 ($tselect = 3$) is supported when Number of Triggers is 4, or
– trigger 3 or 7 ($tselect = 3$ or 7) is supported when Number of Triggers is 8.

### 16.6.6 Trigger Control

**Mnemonic Name:** tcontrol  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug and Machine  
**CSR Address:** 0x7a5 (standard read/write)

This register provides accesses to the `tcontrol` register, and it indicates the current native M-Mode debugging settings.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTE</td>
<td>[3]</td>
<td>M-mode trigger enable field. When a trap into M-mode is taken, MTE is set to 0. When the MRET instruction is executed, MTE is set to the value of MPTE.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MPTE</td>
<td>[7]</td>
<td>M-mode previous trigger enable field. When a trap into M-mode is taken, MPTE is set to the value of MTE.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Triggers do not match/fire while the hart is in M-mode.</td>
</tr>
<tr>
<td>1</td>
<td>Triggers do match/fire while the hart is in M-mode.</td>
</tr>
</tbody>
</table>

### 16.6.7 Machine Context

**Mnemonic Name:** mcontext  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug and Machine  
**CSR Address:** 0x7a8 (standard read/write)
This register provides access to the `mcontext` register.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCONTEXT</td>
<td>[5:0]</td>
<td>Machine mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

### 16.6.8 Supervisor Context

**Mnemonic Name:** scontext  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug, Machine, Supervisor  
**CSR Address:** 0x7aa (standard read/write)

This register provides access to the `scontext` register.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCONTEXT</td>
<td>[8:0]</td>
<td>Machine mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

### 16.6.9 Match Control

**Mnemonic Name:** mcontrol  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug and Machine  
**CSR Address:** 0x7a1 (standard read/write)

This register is accessible as `tdata1` when `TYPE` is 0 or 2.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>[0]</td>
<td>Setting this field to enable this trigger to compare virtual address of a load.</td>
<td>RW*</td>
<td>0</td>
</tr>
<tr>
<td>STORE</td>
<td>[1]</td>
<td>Setting this field to enable this trigger to compare virtual address of a store.</td>
<td>RW*</td>
<td>0</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>[2]</td>
<td>Setting this field to enable this trigger to compare virtual address of an instruction.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>[3]</td>
<td>Setting this field to enable this trigger in U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>[4]</td>
<td>Setting this field to enable this trigger in S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>[6]</td>
<td>Setting this field to enable this trigger in M-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MATCH</td>
<td>[10:7]</td>
<td>Setting this field to select the matching scheme.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Matches when the value equals ( tdata2 ).</td>
</tr>
<tr>
<td>1</td>
<td>Matches when the top ( M ) bits of the value match the top ( M ) bits of ( tdata2 ). ( M ) is 63 minus the index of the least-significant bit containing 0 in ( tdata2 ).</td>
</tr>
<tr>
<td>2</td>
<td>Matches when the value is greater than (unsigned) or equal to ( tdata2 ).</td>
</tr>
<tr>
<td>3</td>
<td>Matches when the value is less than (unsigned) ( tdata2 ).</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAIN</td>
<td>[11]</td>
<td>Setting this field to enable trigger chain.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>When this trigger matches, the configured action is taken.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>While this trigger does not match, it prevents the trigger with the next index from matching.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Number of Triggers is 2, this field is hardwired to 0 on trigger 1 (tselect = 1).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Number of Triggers is 4, this field is hardwired to 0 on trigger 3 (tselect = 3).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Number of Triggers is 8, this field is hardwired to 0 on trigger 3 and trigger 7 (tselect = 3 or 7).</td>
<td></td>
</tr>
<tr>
<td>ACTION</td>
<td>[15:12]</td>
<td>Setting this field to select what happens when this trigger matches.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Raise a breakpoint exception.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enter Debug Mode. (Only supported when DMODE is 1.)</td>
<td></td>
</tr>
<tr>
<td>MASKMAX</td>
<td>[58:53]</td>
<td>Indicates the largest naturally aligned range supported by the hardware is 2^12 bytes.</td>
<td>RO</td>
<td>12</td>
</tr>
<tr>
<td>DMODE</td>
<td>[59]</td>
<td>Setting this field to indicate the trigger is used by Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
### Field Name | Bits | Description | Type | Reset
---|---|---|---|---
TYPE | [63:60] | Indicates the trigger type. | RW | 2

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected trigger is invalid.</td>
</tr>
<tr>
<td>2</td>
<td>The selected trigger is an address/data match trigger.</td>
</tr>
</tbody>
</table>

**Note**

The LOAD/STORE fields take no effect and are cleared if the EXECUTE field is set at the same time.

### 16.6.10 Instruction Count

**Mnemonic Name:** icount  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug and Machine  
**CSR Address:** 0x7a1 (standard read/write)

This register is accessible as tdata1 when TYPE is 3.

This register exists just for single-stepping support so COUNT is hardwired to 1. After this trigger fires, the mode bits the mode bits (M, S, U) will be cleared instead of causing the COUNT bits to be decremented.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DMODE</th>
<th>COUNT</th>
<th>M</th>
<th>S</th>
<th>U</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Field Name | Bits | Description | Type | Reset
---|---|---|---|---
ACTION | [5:0] | Setting this field to select what happens when this trigger matches. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Raise a breakpoint exception.</td>
</tr>
<tr>
<td>1</td>
<td>Enter Debug Mode. (Only supported when DMODE is 1.)</td>
</tr>
</tbody>
</table>

| Field Name | Bits | Description | Type | Reset
---|---|---|---|---
U | [6] | Setting this field to enable this trigger in U-mode. | RW | 0
S | [7] | Setting this field to enable this trigger in S-mode. | RW | 0
M | [9] | Setting this field to enable this trigger in M-mode. | RW | 0

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT</td>
<td>[10]</td>
<td>This field is hardwired to 1 for single-stepping support</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td>DMODE</td>
<td>[59]</td>
<td>Setting this field to indicate the trigger is used by Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td>
</tr>
<tr>
<td>1</td>
<td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td>
</tr>
</tbody>
</table>

| TYPE    | [63:60] | The selected trigger is an instruction count trigger.                      | RW   | 2     |

16.6.11 Interrupt Trigger

Mnemonic Name: itrigger

IM Requirement: DEBUG_SUPPORT

Access Mode: Debug and Machine

CSR Address: 0x7a1 (standard read/write)

This register is accessible as tdata1 when TYPE is 4.

This trigger may fire on any of the interrupts configurable in mie. The interrupts to fire on are configured by setting the same bit in tdata2 as would be set in mie to enable the interrupt.

```
  63  60  59  58                              10  9  8  7  6  5  0
 TYPE DMODE                                  M 0 S U ACTION
```
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>[5:0]</td>
<td>Setting this field to select what happens when this trigger matches.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Raise a breakpoint exception.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enter Debug Mode. (Only supported when DMODE is 1.)</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>[6]</td>
<td>Setting this field to enable this trigger in U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>[7]</td>
<td>Setting this field to enable this trigger in S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>[9]</td>
<td>Setting this field to enable this trigger in M-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>DMODE</td>
<td>[59]</td>
<td>Setting this field to indicate the trigger is used by Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>[63:60]</td>
<td>The selected trigger is an interrupt trigger.</td>
<td>RW</td>
<td>2</td>
</tr>
</tbody>
</table>
16.6.12 Exception Trigger

Mnemonic Name: etrigger
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

This register is accessible as tdata1 when TYPE is 5.

This trigger may fire on up to XLEN of the Exception Codes defined in mcause (with Interrupt=0). Those causes are configured by writing the corresponding bit in tdata2. (E.g. to trap on an illegal instruction, the debugger sets bit 2 in tdata2.)

The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.

---

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>[5:0]</td>
<td>Setting this field to select what happens when this trigger matches.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Raise a breakpoint exception.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enter Debug Mode. (Only supported when DMODE is 1.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>[6]</td>
<td>Setting this field to enable this trigger in U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>[7]</td>
<td>Setting this field to enable this trigger in S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>[9]</td>
<td>Setting this field to enable this trigger in M-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>NMI</td>
<td>[10]</td>
<td>Setting this field to enable this trigger in non-maskable interrupts, regardless of the values of s, u, and m.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMODE</td>
<td>[59]</td>
<td>Setting this field to indicate the trigger is used by Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td>
</tr>
<tr>
<td>1</td>
<td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td>
</tr>
</tbody>
</table>

| TYPE | [63:60] | The selected trigger is an exception trigger. | RW | 2 |
16.6.13 Trigger Extra

Mnemonic Name: textra
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a3 (standard read/write)

This register is accessible as tdata3 when TYPE is 2, 3, 4, or 5 of the currently selected trigger registers selected by the tselect register, and it indicates the context matching scheme of the currently selected trigger.

```
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
| 0 | MVALUE | MSELECT | 0 | SVALUE | SSELECT |
+---+---+---+---+---+---+---+---+---+---+
```

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSELECT</td>
<td>[1:0]</td>
<td></td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Ignore MVALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>This trigger will only match if the lower bits of scontext equal SVALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>This trigger will only match if satp.ASID equals SVALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVALUE</td>
<td>[10:2]</td>
<td>Data used together with SSELECT.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MSELECT</td>
<td>[50]</td>
<td></td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Ignore MVALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>This trigger will only match if the lower bits of mcontext equal MVALUE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVALUE</td>
<td>[56:51]</td>
<td>Data used together with MSELECT.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.7 Debug and Trigger Registers

16.7.1 Debug Control and Status Register

Mnemonic Name: dcsr
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b0 (debug-mode-only)

```
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRV</td>
<td>[1:0]</td>
<td>The privilege level that the hart was operating in when Debug Mode was entered. The external debugger can modify this value to change the hart's privilege level when exiting Debug Mode.</td>
<td>WARL</td>
<td>3</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>User/Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Supervisor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>[2]</td>
<td>This bit controls whether non-Debug Mode instruction execution is in the single step mode. When set, the hart returns to Debug Mode after a single instruction execution. If the instruction does not complete due to an exception, the hart will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Single Step Mode is off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Single Step Mode is on</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMIP</td>
<td>[3]</td>
<td>When this bit is set, there is a Non-Maskable-Interrupt (NMI) pending for the hart. Since an NMI can indicate a hardware error condition, reliable debugging may no longer be possible once this bit becomes set.</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td>MPRVEN</td>
<td>[4]</td>
<td>This bit controls whether mstatus.MPRV takes effect in Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>CAUSE</td>
<td>[8:6]</td>
<td>Reason why Debug Mode was entered. When there are multiple reasons to enter Debug Mode, the priority to determine the CAUSE value will be: trigger module &gt; EBREAK &gt; halt-on-reset &gt; halt request &gt; single step. Halt requests are requests issued by the external debugger.</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MPRV in mstatus is ignored in Debug Mode.</td>
</tr>
<tr>
<td>1</td>
<td>MPRV in mstatus takes effect in Debug Mode.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>EBREAK</td>
</tr>
<tr>
<td>2</td>
<td>Trigger module</td>
</tr>
<tr>
<td>3</td>
<td>Halt request</td>
</tr>
<tr>
<td>4</td>
<td>Single step</td>
</tr>
<tr>
<td>5</td>
<td>Halt-on-reset</td>
</tr>
<tr>
<td>6–7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOPTIME</td>
<td>[9]</td>
<td>This bit controls whether timers are stopped in Debug Mode. The processor only drives its stoptime output pin to 1 if it is in Debug Mode and this bit is set. Integration effort is required to make timers in the platform observe this pin to really stop them.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>STOPCOUNT</td>
<td>[10]</td>
<td>This bit controls whether performance counters are stopped in Debug Mode.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>STEPIE</td>
<td>[11]</td>
<td>This bit controls whether interrupts are enabled during single stepping.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>EBREAKU</td>
<td>[12]</td>
<td>This bit controls the behavior of EBREAK instructions in User/Application Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

### STOPTIME

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not stop timers in Debug Mode</td>
</tr>
<tr>
<td>1</td>
<td>Stop timers in Debug Mode</td>
</tr>
</tbody>
</table>

### STOPCOUNT

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not stop counters in Debug Mode</td>
</tr>
<tr>
<td>1</td>
<td>Stop counters in Debug Mode</td>
</tr>
</tbody>
</table>

### STEPIE

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable interrupts during single stepping</td>
</tr>
<tr>
<td>1</td>
<td>Allow interrupts in single stepping</td>
</tr>
</tbody>
</table>

### EBREAKU

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Generate a regular breakpoint exception</td>
</tr>
<tr>
<td>1</td>
<td>Enter Debug Mode</td>
</tr>
</tbody>
</table>

Continued on next page...
### Field Name Bits Description Type Reset

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBREAKS</td>
<td>[13]</td>
<td>This bit controls the behavior of EBREAK instructions in Supervisor Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Generate a regular breakpoint exception</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enter Debug Mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBREAKM</td>
<td>[15]</td>
<td>This bit controls the behavior of EBREAK instructions in Machine Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Generate a regular breakpoint exception</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enter Debug Mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XDEBUGVER</td>
<td>[31:28]</td>
<td>Version of the external debugger. 0 indicates that no external debugger exists and 4 indicates that the external debugger conforms to the RISC-V External Debug Support (TD003) V0.13.</td>
<td>RO</td>
<td>4</td>
</tr>
</tbody>
</table>

#### 16.7.2 Debug Program Counter

**Mnemonic Name:** dpc  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug  
**CSR Address:** 0x7b1 (debug-mode-only)

When entering Debug Mode, the dpc CSR is updated with the virtual address of the next instruction to be executed. The behavior is described in more detail in Table 55. When leaving Debug Mode, the hart's PC is updated to the value stored in this register. The external debugger may write this register to change where the hart resumes.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPC</td>
<td>[63:0]</td>
<td>Debug Program Counter. Bit 0 is hardwired to 0.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 55: Virtual Address in DPC upon Debug Mode Entry

<table>
<thead>
<tr>
<th>Cause</th>
<th>Virtual Address in DPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBREAK</td>
<td>Address of the EBREAK instruction</td>
</tr>
<tr>
<td>single step</td>
<td>Address of the instruction that would be executed next if no debugging was going on.</td>
</tr>
<tr>
<td>trigger module</td>
<td>Address of the instruction which caused the trigger module to fire.</td>
</tr>
<tr>
<td>halt request</td>
<td>Address of the next instruction to be executed at the time that Debug Mode was entered</td>
</tr>
</tbody>
</table>

16.7.3 Debug Scratch Register 0

Mnemonic Name: dscratch0
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b2 (debug-mode-only)

![DSCRATCH0]

A scratch register that is reserved for use by Debug Module.

16.7.4 Debug Scratch Register 1

Mnemonic Name: dscratch1
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b3 (debug-mode-only)

![DSCRATCH1]

A scratch register that is reserved for use by Debug Module.

16.7.5 Exception Redirection Register

Mnemonic Name: dexc2dbg
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7e0 (non-standard read/write)
This register redirects selected exceptions to cause the hart to enter Debug Mode instead of performing the standard trap handling.

When an exception is redirected to enter Debug Mode, the \texttt{dpc} CSR will be updated with the virtual address of the instruction causing the exception. The \texttt{dcsr.CAUSE} field will be updated with a value of 1 (EBREAK). The actual cause of the exception is saved to the \texttt{ddcause} CSR. The required updates to \texttt{mepc}, \texttt{mcause}, \texttt{mtval}, \texttt{mstatus}, and \texttt{mxstatus} CSRs for exceptions will not be affected by the redirection and these CSRs continue to provide information associated with the corresponding exceptions.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM</td>
<td>[0]</td>
<td>Indicates whether Instruction Access Misaligned exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>IAF</td>
<td>[1]</td>
<td>Indicates whether Instruction Access Fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td>[2]</td>
<td>Indicates whether Illegal Instruction exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>NMI</td>
<td>[3]</td>
<td>Indicates whether Non-Maskable Interrupt exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td>1</td>
<td>Redirect</td>
</tr>
</tbody>
</table>

### Field Name Bits Description

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM</td>
<td>[0]</td>
<td>Indicates whether Instruction Access Misaligned exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>IAF</td>
<td>[1]</td>
<td>Indicates whether Instruction Access Fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td>[2]</td>
<td>Indicates whether Illegal Instruction exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>NMI</td>
<td>[3]</td>
<td>Indicates whether Non-Maskable Interrupt exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td>1</td>
<td>Redirect</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAM</td>
<td>[4]</td>
<td>Indicates whether Load Access Misaligned exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
<tr>
<td>LAF</td>
<td>[5]</td>
<td>Indicates whether Load Access Fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>[6]</td>
<td>Indicates whether Store Access Misaligned exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
<tr>
<td>SAF</td>
<td>[7]</td>
<td>Indicates whether Store Access Fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
<tr>
<td>UEC</td>
<td>[8]</td>
<td>Indicates whether U-mode Environment Call exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>[9]</td>
<td>Indicates whether S-mode Environment Call exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEC</td>
<td>[11]</td>
<td>Indicates whether M-mode Environment Call exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
</tr>
<tr>
<td>HSP</td>
<td>[12]</td>
<td>Indicates whether Stack Protection exceptions are redirected to enter Debug Mode. This bit is present only when \texttt{mmsc_cfg.HSP} is set.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
</tr>
<tr>
<td>SLPECC</td>
<td>[14]</td>
<td>Indicates whether local memory slave port ECC Error local interrupts are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
</tr>
<tr>
<td>BWE</td>
<td>[15]</td>
<td>Indicates whether Bus-write Transaction Error local interrupts are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
</tr>
<tr>
<td>IPF</td>
<td>[16]</td>
<td>Indicates whether instruction page fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Redirect</td>
</tr>
</tbody>
</table>

Continued on next page...
### Field Name Bits Description Type Reset

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPF</td>
<td>[17]</td>
<td>Indicates whether load fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>SPF</td>
<td>[18]</td>
<td>Indicates whether store page fault exceptions are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>PMOV</td>
<td>[19]</td>
<td>Indicates whether performance counter overflow interrupts are redirected to enter Debug Mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Value Meaning

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not redirect</td>
</tr>
<tr>
<td>1</td>
<td>Redirect</td>
</tr>
</tbody>
</table>

**16.7.6 Debug Detailed Cause**

**Mnemonic Name:** ddcause  
**IM Requirement:** DEBUG_SUPPORT  
**Access Mode:** Debug  
**CSR Address:** 0x7e1 (non-standard read/write)

```
<table>
<thead>
<tr>
<th></th>
<th>SUBTYPE</th>
<th>MAINTYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINTYPE</td>
<td>[7:0]</td>
<td>Cause for redirection to Debug Mode.</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Software Breakpoint (EBREAK)</td>
</tr>
<tr>
<td>1</td>
<td>Instruction Access Misaligned (IAM)</td>
</tr>
<tr>
<td>2</td>
<td>Instruction Access Fault (IAF)</td>
</tr>
<tr>
<td>3</td>
<td>Illegal Instruction (II)</td>
</tr>
<tr>
<td>4</td>
<td>Non-Maskable Interrupt (NMI)</td>
</tr>
<tr>
<td>5</td>
<td>Load Access Misaligned (LAM)</td>
</tr>
<tr>
<td>6</td>
<td>Load Access Fault (LAF)</td>
</tr>
<tr>
<td>7</td>
<td>Store Access Misaligned (SAM)</td>
</tr>
<tr>
<td>8</td>
<td>Store Access Fault (SAF)</td>
</tr>
<tr>
<td>9</td>
<td>U-mode Environment Call (UEC)</td>
</tr>
<tr>
<td>10</td>
<td>S-mode Environment Call (SEC)</td>
</tr>
<tr>
<td>11</td>
<td>Instruction page fault</td>
</tr>
<tr>
<td>12</td>
<td>M-mode Environment Call (MEC)</td>
</tr>
<tr>
<td>13</td>
<td>Load page fault</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Store/AMO page fault</td>
</tr>
<tr>
<td>16</td>
<td>Imprecise ECC error</td>
</tr>
<tr>
<td>17</td>
<td>Bus write transaction error</td>
</tr>
<tr>
<td>18</td>
<td>Performance Counter overflow</td>
</tr>
<tr>
<td>19–31</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Stack overflow exception</td>
</tr>
<tr>
<td>33</td>
<td>Stack underflow exception</td>
</tr>
<tr>
<td>≥34</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBTYPE</td>
<td>[15:8]</td>
<td>Subtypes for main type. The table below lists the subtypes for DCSR.CAUSE==1 and DDCAUSE.MAINTYPE==3.</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>1</td>
<td>Privileged instruction</td>
</tr>
<tr>
<td>2</td>
<td>Non-existent CSR</td>
</tr>
<tr>
<td>3</td>
<td>Privilege CSR access</td>
</tr>
<tr>
<td>4</td>
<td>Read-only CSR update</td>
</tr>
</tbody>
</table>
16.8 Supervisor Trap Related CSRs

16.8.1 Supervisor Status

Mnemonic Name: sstatus
IM Requirement: misa[18]==1
Access Mode: Supervisor
CSR Address: 0x100 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIE</td>
<td>[0]</td>
<td>U-mode interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIE</td>
<td>[1]</td>
<td>S-mode interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPIE</td>
<td>[4]</td>
<td>UPIE holds the value of the UIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIE</td>
<td>[5]</td>
<td>SPIE holds the value of the SIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPP</td>
<td>[8]</td>
<td>SPP holds the privilege mode prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encoding is 1 for S-mode and 0 for U-mode.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
FS holds the status of the architectural states of the floating-point unit, including the fcsr CSR and f0 – f31 floating-point data registers. The value of this field is zero and read-only if the processor does not have FPU. This field is primarily managed by software. The processor hardware assists the state managements in two regards:

- Attempts to access fcsr or any f register raise an illegal-instruction exception when FS is Off.
- Otherwise, FS is updated to the Dirty state by any instruction that updates fcsr or any f register.

Changing the setting of this field has no effect on the contents of the floating-point register states. In particular, setting FS to Off does not destroy the states, nor does setting FS to Initial clear the contents.

The same copy of FS bits are shared by both mstatus and sstatus. Normally the supervisor mode privileged software would use the FS bits to manage deferred context switches of FPU states. Machine mode software should be more conservative in managing context switches using FS bits.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Initial</td>
</tr>
<tr>
<td>2</td>
<td>Clean</td>
</tr>
<tr>
<td>3</td>
<td>Dirty</td>
</tr>
</tbody>
</table>

Continued on next page...
SUM [18] SUM controls whether a S-mode load/store instruction to a user accessible page is allowed or not when page translation is enabled. It is in effect in two scenarios: (a) M-mode with MPRV=1 and MPP=S, and (b) in S-mode. It has no effect when page-based virtual memory is not in effect. A page is user accessible when the U bit of the corresponding PTE entry is 1.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>1</td>
<td>Allowed</td>
</tr>
</tbody>
</table>

MXR [19] MXR controls whether execute-only pages are readable. It has no effect when page-based virtual memory is not in effect.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execute-only pages are not readable</td>
</tr>
<tr>
<td>1</td>
<td>Execute-only pages are readable</td>
</tr>
</tbody>
</table>

UXL [33:32] UXL controls the value of XLEN for U-mode. When U-mode is not available, this field is hardwired to 0.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
</tbody>
</table>

SD [63] SD summarizes whether either the FS field or XS field is dirty.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When N extension is not supported, the corresponding bits in sstatus are hardwired to zero.
16.8.2 Supervisor Exception Delegation

Mnemonic Name: sedeleg
Access Mode: Supervisor
CSR Address: 0x102 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM</td>
<td>[0]</td>
<td>IAM indicates whether an Instruction Address Misaligned exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value:</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>IAF</td>
<td>[1]</td>
<td>IAF indicates whether an Instruction Access Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value:</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>[2]</td>
<td>II indicates whether an Illegal Instruction exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value:</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>[3]</td>
<td>B indicates whether an exception triggered by breakpoint will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value:</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAM</td>
<td>[4]</td>
<td>LAM indicates whether a Load Address Misaligned exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>LAF</td>
<td>[5]</td>
<td>LAF indicates whether a Load Access Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>[6]</td>
<td>SAM indicates whether a Store/AMO Address Misaligned exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SAF</td>
<td>[7]</td>
<td>SAF indicates whether a Store/AMO Access Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>UEC</td>
<td>[8]</td>
<td>UEC indicates whether an exception triggered by environment call from U-mode will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPF</td>
<td>[12]</td>
<td>IPF indicates whether an Instruction Page Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>LPF</td>
<td>[13]</td>
<td>LPF indicates whether a Load Page Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
<tr>
<td>SPF</td>
<td>[15]</td>
<td>SPF indicates whether a Store/AMO Page Fault exception will be delegated to U-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not delegate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>delegate</td>
<td></td>
</tr>
</tbody>
</table>

When N extension is not supported, the corresponding bits in `sedeleg` are hardwired to zero.
16.8.3 Supervisor Interrupt Delegation

Mnemonic Name: sideleg
Access Mode: Supervisor
CSR Address: 0x103 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USI</td>
<td>[0]</td>
<td>USI indicates whether an U-mode software interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UTI</td>
<td>[4]</td>
<td>UTI indicates whether an U-mode timer interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UEI</td>
<td>[8]</td>
<td>UEI indicates whether an U-mode external interrupt will be delegated to S-mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

When N extension is not supported, the corresponding bits in sideleg are hardwired to zero.
16.8.4 Supervisor Interrupt Enable

Mnemonic Name: sie
IM Requirement: misa[18] = 1
Access Mode: Supervisor
CSR Address: 0x104 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIE</td>
<td>[0]</td>
<td>U-mode software interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| SSIE       | [1]  | S-mode software interrupt enable bit. | RW   | 0     |
| Value      | Meaning |
| 0          | Disabled |
| 1          | Enabled  |

| UTIE       | [4]  | U-mode timer interrupt enable bit. | RW   | 0     |
| Value      | Meaning |
| 0          | Disabled |
| 1          | Enabled  |

| STIE       | [5]  | S-mode timer interrupt enable bit. | RW   | 0     |
| Value      | Meaning |
| 0          | Disabled |
| 1          | Enabled  |

| UEIE       | [8]  | U-mode external interrupt enable bit. | RW   | 0     |
| Value      | Meaning |
| 0          | Disabled |
| 1          | Enabled  |

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEIE</td>
<td>[9]</td>
<td>S-mode external interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

When N extension is not supported, the corresponding bits in `sie` are hardwired to zero.
16.8.5 Supervisor Trap Vector Base Address

Mnemonic Name: stvec
IM Requirement: misa[18] = = 1
Access Mode: Supervisor
CSR Address: 0x105 (standard read/write)

This register determines the base address of the trap vector for S-mode trap handling. The least significant 2 bits are hardwired to zeros. When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

When \texttt{mmisc_ctl.VEC_PLIC} is 0 (PLIC is not in the vector mode), this register indicates the entry points for the trap handler and it may point to any 4-byte aligned location in the memory space.

On the other hand, when \texttt{mmisc_ctl.VEC_PLIC} is 1 (PLIC is in the vector mode), this register will be the base address of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.

- This register should be aligned to $2^{\log_2 N+2}$-byte boundary for PLIC with \(N\) interrupt sources. For example, if \(N\) is 1023, the minimum alignment requirement is 4096 bytes (4 KiB).
- \texttt{stvec[0]} is for exceptions and non-external local interrupts.
- \texttt{stvec[i]} is for external PLIC interrupt source \(i\) triggered through the \texttt{sip.SEIP} pending condition when \texttt{mideleg.SEI == 1}.
- \texttt{stvec[1024+i]} is for external PLIC interrupt source \(i\) triggered through the \texttt{sip.UEIP} pending condition when \texttt{sideleg.UEI == 0}.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE[63:2]</td>
<td>[63:2]</td>
<td>Base address for interrupt and exception handlers. See description above for alignment requirements when PLIC is in the vector mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.8.6 Supervisor Counter Enable Register

Mnemonic Name: scounteren
IM Requirement: misa[18] = =1
Access Mode: Supervisor
CSR Address: 0x106 (non-standard read/write)

The supervisor counter-enable register controls the availability of the hardware performance monitoring counters to U-mode.

If S-mode is not permitted to access a counter register or when the corresponding bit in the scounteren register is zero, attempts to read the corresponding register while executing in U-mode will cause an illegal instruction exception. If S-mode is permitted to access a counter register and the corresponding bit is set, access to the counter register is permitted in U-mode.

In summary, a counter can be accessed in U-mode only when (mcounteren.counter==1) and (scounteren.counter==1).
16.8.7 Supervisor Scratch Register

Mnemonic Name: sscratch
IM Requirement: misa[18] = 1
Access Mode: Supervisor
CSR Address: 0x140 (standard read/write)

A scratch register for temporary data storage, which is typically used by the S-mode trap handler.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCRATCH</td>
<td>[63:0]</td>
<td>Scratch register storage.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.8.8 Supervisor Exception Program Counter

Mnemonic Name: sepc
IM Requirement: misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x141 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC</td>
<td>[63:1]</td>
<td>Exception program counter.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

This register is written with the virtual address of the instruction that raises a trap and the trap is taken to S-mode.
16.8.9 Supervisor Cause Register

Mnemonic Name: scause
IM Requirement: misa[18] = = 1
Access Mode: Supervisor
CSR Address: 0x142 (standard read/write)

This register indicates the cause of traps when they are taken to S-mode.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTION_CODE</td>
<td>[9:0]</td>
<td>Exception Code.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>[63]</td>
<td>Interrupt.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Each local interrupt can be configured with a local interrupt number. For S-mode local interrupts, cause numbers will be (local interrupt number + 256). Cause numbers below show the default cause numbers of local interrupts.

Table 56: AX45MP-1C scause Value After Trap

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>User software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>User timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>User external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Supervisor external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>256+16</td>
<td>Slave port ECC error interrupt (S-mode)</td>
</tr>
<tr>
<td>1</td>
<td>256+17</td>
<td>Bus write transaction error interrupt (S-mode)</td>
</tr>
<tr>
<td>1</td>
<td>256+18</td>
<td>Performance monitor overflow interrupt(S-mode)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>Environment call from S-mode</td>
</tr>
<tr>
<td>0</td>
<td>11:10</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>Instruction page fault</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>Load page fault</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>Store/AMO page fault</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>Stack overflow exception</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>Stack underflow exception</td>
</tr>
<tr>
<td>0</td>
<td>40-47</td>
<td>Andes Custom Extension exception (see Andes Custom Extension Specification CUM009 for more details)</td>
</tr>
</tbody>
</table>
16.8.10 Supervisor Trap Value

Mnemonic Name: stval
IM Requirement: misa[18] = =1
Access Mode: Supervisor
CSR Address: 0x143 (standard read/write)

This register is updated when a trap is taken to S-mode. The updated value is dependent on the cause of traps:

- For Hardware Breakpoint exceptions, Address Misaligned exceptions, Access Fault exceptions, or Page Fault exceptions, it is the effective faulting addresses.

- For illegal instruction exceptions, the updated value is the faulting instruction. If the length of the instruction is less than XLEN bits long, the upper bits of stval are cleared to zero.

- For other exceptions, stval is set to zero.

For instruction-fetch access faults, this register will be updated with the address pointing to the portion of the instruction that caused the fault, while the sepc register will be updated with the address pointing to the beginning of the instruction.

When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>STVAL</td>
<td>[63:0]</td>
<td>Exception-specific information for software trap handling.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
## 16.8.11 Supervisor Interrupt Pending

**Mnemonic Name:** sip  
**IM Requirement:** misa[18] = 1  
**Access Mode:** Supervisor  
**CSR Address:** 0x144 (standard read/write)

### CSR Address

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEIP</td>
<td>UEIP</td>
<td>0</td>
<td>STIP</td>
<td>UTIP</td>
<td>0</td>
<td>SSIP</td>
<td>USIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Field Name | Bits | Description | Type | Reset
---|---|---|---|---
USIP | [0] | U-mode software interrupt pending bit. | RW | 0
| **Value** | **Meaning** |
| 0 | Not pending |
| 1 | Pending |

SSIP | [1] | S-mode software interrupt pending bit. | RW | 0
| **Value** | **Meaning** |
| 0 | Not pending |
| 1 | Pending |

UTIP | [4] | U-mode timer interrupt pending bit. | RO | 0
| **Value** | **Meaning** |
| 0 | Not pending |
| 1 | Pending |

STIP | [5] | S-mode timer interrupt pending bit. | RO | 0
| **Value** | **Meaning** |
| 0 | Not pending |
| 1 | Pending |

UEIP | [8] | U-mode external interrupt pending bit. | RW | 0
| **Value** | **Meaning** |
| 0 | Not pending |
| 1 | Pending |

---

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEIP</td>
<td>[9]</td>
<td>S-mode external interrupt pending bit.</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not pending</td>
</tr>
<tr>
<td>1</td>
<td>Pending</td>
</tr>
</tbody>
</table>

When N extension is not supported, the corresponding bits in sip are hardwired to zero.
16.8.12 Supervisor Local Interrupt Enable

Mnemonic Name: slie
IM Requirement: misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9C4 (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMECCI</td>
<td>[16]</td>
<td>Enable S-mode imprecise ECC error local interrupt. The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>BWEI</td>
<td>[17]</td>
<td>Enable S-mode bus read/write transaction error local interrupt. The processor may receive bus errors on load/store instructions or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

If a supervisor local interrupt is enabled, the supervisor local interrupt can be taken in the mode depending on the mslideleg CSR. When mslideleg for a local interrupt is set, the supervisor local interrupt will be served in S-mode. Otherwise, it will be served in M-mode.

The privileged mode of IMECCI and BWEI are determined by the current privileged mode. For M/S/U configuration, if the current privileged mode is User or Supervisor, the local interrupt generated is a supervisor local interrupt. For M/U configuration, if the current privileged mode is User, the local interrupt generated is a machine local interrupt. The privileged mode of the above PMOVI interrupt is determined by the counter state of mcountermask_m.

Each local interrupt can be configured with a local interrupt interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields below show the default bit location.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMOVI</td>
<td>[18]</td>
<td>Enable S-mode performance monitor overflow local interrupt.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Local interrupt is not enabled.</td>
</tr>
<tr>
<td>1</td>
<td>Local interrupt is enabled.</td>
</tr>
</tbody>
</table>
16.8.13 Supervisor Local Interrupt Pending

Mnemonic Name: slip
IM Requirement: misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9C5 (non-standard read/write)

This register indicates whether a supervisor local interrupt is pending or not. If an enabled supervisor local interrupt is pending, the supervisor local interrupt will be taken in the mode depending on the mslideleg CSR. When mslideleg for a local interrupt is set, the supervisor local interrupt will be served in S-mode. Otherwise, it will be served in M-mode.

The privileged mode of IMECCI and BWEI are determined by the current privileged mode. For M/S/U configuration, if the current privileged mode is User or Supervisor, the local interrupt generated is a supervisor local interrupt. For M/U configuration, if the current privileged mode is User, the local interrupt generated is a machine local interrupt. The privileged mode of the above PMOVI interrupt is determined by the counter state of mcountermask_m.

Each local interrupt can be configured with a local interrupt interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields below show the default bit location.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMECCI</td>
<td>[16]</td>
<td>Pending status of S-mode imprecise ECC error local interrupt. The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Local interrupt is not pending.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Local interrupt is pending.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWEI</td>
<td>[17]</td>
<td>Pending status of S-mode bus read/write transaction error local interrupt. The processor may receive bus errors on load/store instructions or cache writebacks.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Local interrupt is not pending.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Local interrupt is pending.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMOVI</td>
<td>[18]</td>
<td>Pending status of S-mode performance monitor overflow local interrupt.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Local interrupt is not pending.</td>
</tr>
<tr>
<td>1</td>
<td>Local interrupt is pending.</td>
</tr>
</tbody>
</table>
16.8.14 Supervisor Detailed Trap Cause

Mnemonic Name: sdcause
IM Requirement: Required if supervisor mode is implemented
Access Mode: Supervisor
CSR Address: 0x9c9 (non-standard read/write)

For precise exceptions:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SDCAUSE</td>
</tr>
</tbody>
</table>

For imprecise exceptions (local interrupts):

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>PM</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SDCAUSE</td>
</tr>
</tbody>
</table>

When multiple events cause traps to be taken with the same scause value, this register helps to further disambiguate them. Some events could cause either precise exceptions or imprecise exceptions (local interrupts) depending on when they are detected, so they can appear in multiple tables below.

Imprecise exceptions are triggered as local interrupts, so the tables below for scause == Local Interrupt n summarizes imprecise exceptions delivered as local interrupt n. The scause == Local Interrupt n notation stands for (INTERRUPT, EXCEPTION_CODE) fields of scause is (1, n).

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDCAUSE</td>
<td>[2:0]</td>
<td>This register further disambiguates causes of traps recorded in the scause register. See the list below for details.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>PM</td>
<td>[6:5]</td>
<td>When scause is imprecise exception (in the form of an interrupt), the PM field records the privileged mode of the instruction that caused the imprecise exception. The PM field encoding is defined as follows:</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User mode</td>
</tr>
<tr>
<td>1</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Machine mode</td>
</tr>
</tbody>
</table>

The value of SDCAUSE for precise exception:
• When `scause == 1` (Instruction access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP instruction access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>PMA empty hole access</td>
</tr>
</tbody>
</table>

• When `scause == 2` (Illegal instruction)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Please parse the <code>stval</code> CSR</td>
</tr>
<tr>
<td>1</td>
<td>FP disabled exception</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

• When `scause == 5` (Load access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP load access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>5</td>
<td>PMA empty hole access</td>
</tr>
<tr>
<td>6</td>
<td>PMA attribute inconsistency</td>
</tr>
<tr>
<td>7</td>
<td>PMA NAMO exception</td>
</tr>
</tbody>
</table>

• When `scause == 7` (Store access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP store access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
</tbody>
</table>

Continued on next page…
The value of SDCAUSE for imprecise exception:

- When `scause` = = *Local Interrupt 272 (16 + 256)* (ECC error local interrupt)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>LM slave port ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>Imprecise store ECC/Parity error</td>
</tr>
<tr>
<td>3</td>
<td>Imprecise load ECC/Parity error</td>
</tr>
</tbody>
</table>

- When `scause` = = *Local Interrupt 273 (17 + 256)* (Bus read/write transaction error local interrupt)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Bus read error</td>
</tr>
<tr>
<td>2</td>
<td>Bus write error</td>
</tr>
<tr>
<td>3</td>
<td>PMP error caused by load instructions</td>
</tr>
<tr>
<td>4</td>
<td>PMP error caused by store instructions</td>
</tr>
<tr>
<td>5</td>
<td>PMA error caused by load instructions&quot;</td>
</tr>
<tr>
<td>6</td>
<td>PMA error caused by store instructions</td>
</tr>
</tbody>
</table>

- For PMOVI, SDCAUSE will be written 0. For other exceptions and interrupts, this register will not be updated.
16.8.14.1 Detailed Exception Priority

Within Instruction/Load/Store access fault exceptions, the priority of a PMP exception is higher than the priority of a PMA exception, when both types of exceptions happen on the same instruction.
16.9 Supervisor Translation Related CSRs

16.9.1 Supervisor Address Translation and Protection

Mnemonic Name: satp
IM Requirement: misa[18]==1
Access Mode: Supervisor
CSR Address: 0x180 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td>[43:0]</td>
<td>PPN holds the physical page number of the root page table.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>ASID</td>
<td>[52:44]</td>
<td>ASID holds the address space identifier.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>MODE</td>
<td>[63:60]</td>
<td>MODE holds the page translation mode. When MODE is Bare, virtual addresses are equal to physical addresses in S-mode. When MMU is not supported in the product, this CSR will be hardwired to 0.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bare</td>
<td>No page translation</td>
</tr>
<tr>
<td>8</td>
<td>Sv39</td>
<td>Page-based 39-bit virtual addressing</td>
</tr>
<tr>
<td>9</td>
<td>Sv48</td>
<td>Page-based 48-bit virtual addressing</td>
</tr>
</tbody>
</table>
16.10 Supervisor Counter Related CSRs

16.10.1 Supervisor Counter Mask for Machine Mode

Mnemonic Name: scountermask_m
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9D1 (non-standard read/write)

The supervisor counter mask for M-mode register is used to disable M-mode counting for each counter.

This register is an alias of the mcountermask_m CSR, and its default value is 0. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read.

Each bit of this register is writable if the corresponding bit in the mcounterwen CSR is 1.

When the CY, IR, HPM3, HPM4, HPM5, or HPM6 in the scountermask_m register is set, the specific counter will not be incremented in M-mode.
16.10.2 Supervisor Counter Mask for Supervisor Mode

Mnemonic Name: scountermask_s
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9D2 (non-standard read/write)

The supervisor counter mask for S-mode register controls the performance counter behavior in S-mode. The default value of this register is 0.

This register is an alias of the mcountermask_s CSR, and its default value is 0. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read.

Each bit of this register is writable if the corresponding bit in the mcounterwen CSR is 1.

When the CY, IR, HPM3, HPM4, HPM5, or HPM6 in the scountermask_s register is set, the specific counter will not be incremented in S-mode.
16.10.3 Supervisor Counter Mask for User Mode

Mnemonic Name: scountermask_u
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9D3 (non-standard read/write)

The supervisor counter mask for U-mode register controls the performance counter behavior in U-mode. The default value of this register is 0.

This register is an alias of the mcountermask_u CSR, and its default value is 0. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read.

Each bit of this register is writable if the corresponding bit in the mcounterwen CSR is 1.

When the CY, IR, HPM3, HPM4, HPM5, or HPM6 in the scountermask_u register is set, the specific counter will not be incremented in S-mode.
16.10.4 Supervisor Counter Interrupt Enable

Mnemonic Name: scounterinten
IM Requirement: mmsc_cfg.PMNDS==1 & misa[18]==1
Access Mode: Supervisor
CSR Address: 0x9CF (non-standard read/write)

The supervisor counter interrupt enable register controls whether a counter overflow interrupt is generated or not. This register is an alias of the mcounterinten CSR, and its default value is 0. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read.

Each bit of this register is writable if the corresponding bit in the mcounterwen CSR is 1.

When the CY, IR, HPM3, HPM4, HPM5, or HPM6 in this register is 0, no overflow interrupt is generated for the corresponding counter. When one of these bits is set, an interrupt will be generated when the corresponding counter overflows (the counter value wraps around back to 0).
16.10.5 Supervisor Counter Overflow Status

Mnemonic Name: scounterovf
IM Requirement: mmsc_cfg.PMNDs == 1 & misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9D4 (non-standard read/write)

The supervisor counter overflow status register records the overflow status of each counter. When a bit is set, it indicates that an overflow has happened to the corresponding counter. This register is an alias of the mcounterovf CSR. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read. If the corresponding bit in the mcounterwen CSR is 1, writing 0 will deassert the bit and writing 1 will have no effect.
16.10.6 Supervisor Counter-Inhibit

Mnemonic Name: scountinhibit
IM Requirement: mmsc_cfg.PMNDS == 1 & misa[18] == 1
Access Mode: Supervisor
CSR Address: 0x9E0 (non-standard read/write)

This register is an alias of the mcountinhibit CSR. Each bit of this register is read-only if the corresponding bit in the mcounterwen CSR is 0. Writing a read-only bit of this register with a CSR write instruction will not generate any exception. This register is not controlled by the mcounteren register and can always be read.

Each bit of this register is writable if the corresponding bit in the mcounterwen CSR is 1.
16.10.7 Supervisor Performance Monitoring Event Selector

Mnemonic Name: shpmevent3–shpmevent6  
IM Requirement: mmsc_cfg.PMNDS == 1 & misa[18] == 1  
Access Mode: Supervisor  
CSR Address: 0x9E3 to 0x9E6 (non-standard read/write)

The monitoring event is defined in Section 16.4.5.

The read behavior of this register is controlled by the mcounteren register. The write behavior of this register is controlled by the mcounterwen register.
16.11 User Trap Related CSRs

16.11.1 User Status

Mnemonic Name: ustatus
IM Requirement: misa[13] == 1
Access Mode: User
CSR Address: 0x000 (standard read/write)

![Field Name Bits Description Type Reset]
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIE</td>
<td>[0]</td>
<td>U-mode interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPIE</td>
<td>[4]</td>
<td>UPIE holds the value of the UIE bit prior to a trap.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
### 16.11.2 User Interrupt Enable

**Mnemonic Name:** uie  
**IM Requirement:** misa[13] = 1  
**Access Mode:** User  
**CSR Address:** 0x004 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIE</td>
<td>[0]</td>
<td>U-mode software interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UTIE</td>
<td>[4]</td>
<td>U-mode timer interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UEIE</td>
<td>[8]</td>
<td>U-mode external interrupt enable bit.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

#### USB

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
16.11.3 User Trap Vector Base Address

Mnemonic Name: utvec
IM Requirement: misa[13] = 1
Access Mode: User
CSR Address: 0x005 (standard read/write)

This register determines the base address of the trap vector for U-mode trap handling. The least significant 2 bits are hardwired to zeros. When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

When mmisc_ctl.VEC_PLIC is 0 (PLIC is not in the vector mode), this register indicates the entry points for the trap handler and it may point to any 4-byte aligned location in the memory space.

On the other hand, when mmisc_ctl.VEC_PLIC is 1 (PLIC is in the vector mode), this register will be the base address of a vector table with 4-byte entries storing addresses pointing to interrupt service routines. And this register should be aligned to $2^{\log_2 N + 2}$-byte boundary for PLIC with $N$ interrupt sources. For example, if $N$ is 1023, the minimum alignment requirement is 4096 bytes (4 KiB).

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE[63:2]</td>
<td>[63:2]</td>
<td>Base address for interrupt and exception handlers. See description above for alignment requirements when PLIC is in the vector mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.11.4 User Scratch Register

Mnemonic Name: uscratch  
IM Requirement: misa[13] == 1  
Access Mode: User  
CSR Address: 0x040 (standard read/write)

A scratch register for temporary data storage, which is typically used by the U-mode trap handler.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USCRATCH</td>
<td>[63:0]</td>
<td>Scratch register storage.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.11.5 User Exception Program Counter

Mnemonic Name: uepc
IM Requirement: misa[13] = = 1
Access Mode: User
CSR Address: 0x041 (standard read/write)

This register is written with the virtual address of the instruction that raises a trap and the trap is taken to U-mode.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC</td>
<td>[63:1]</td>
<td>Exception program counter.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
16.11.6 User Cause Register

Mnemonic Name: ucause
IM Requirement: misa[13] = = 1
Access Mode: User
CSR Address: 0x042 (standard read/write)

This register indicates the cause of traps when they are taken to U-mode.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTION_CODE</td>
<td>[9:0]</td>
<td>Exception Code.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>[63]</td>
<td>Interrupt.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 57: AX45MP-1C ucause Value After Trap

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>User software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>User timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>User external interrupt</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9-11</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>Instruction page fault</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>Load page fault</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>Store/AMO page fault</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>Stack overflow exception</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33</td>
<td>Stack underflow exception</td>
</tr>
<tr>
<td>0</td>
<td>40-47</td>
<td>Andes Custom Extension exception (see <em>Andes Custom Extension Specification CUM009</em> for more details)</td>
</tr>
</tbody>
</table>
16.11.7 User Trap Value

Mnemonic Name: utval
IM Requirement: misa[13] = =1
Access Mode: User
CSR Address: 0x043 (standard read/write)

This register is updated when a trap is taken to U-mode. The updated value is dependent on the cause of traps:

- For hardware breakpoint exceptions, address-misaligned exceptions, access-fault exceptions, or page-fault exceptions, it is the effective faulting addresses.
- For illegal instruction exceptions, the updated value is the faulting instruction. If the length of the instruction is less than XLEN bits long, the upper bits of utval are cleared to zero.
- For other exceptions, utval is set to zero.

For instruction-fetch access faults, this register will be updated with the address pointing to the portion of the instruction that caused the fault, while the sepc register will be updated with the address pointing to the beginning of the instruction.

When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTVAL</td>
<td>[63:0]</td>
<td>Exception-specific information for software trap handling.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
### 16.11.8 User Interrupt Pending

**Mnemonic Name:** uip  
**IM Requirement:** misa[13] = = 1  
**Access Mode:** User  
**CSR Address:** 0x044 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIP</td>
<td>[0]</td>
<td>U-mode software interrupt pending bit.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Not pending</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Pending</td>
<td></td>
</tr>
</tbody>
</table>

| UTIP       | [4]  | U-mode timer interrupt pending bit. | RO   | 0     |
|            |      | **Value**   | **Meaning** |       |
|            |      | 0           | Not pending |       |
|            |      | 1           | Pending    |       |

| UEIP       | [8]  | U-mode external interrupt pending bit. | RO   | 0     |
|            |      | **Value**   | **Meaning** |       |
|            |      | 0           | Not pending |       |
|            |      | 1           | Pending    |       |
16.11.9 User Detailed Trap Cause

**Mnemonic Name:** udcause
**IM Requirement:** misa[13] == 1
**Access Mode:** User
**CSR Address:** 0x809 (non-standard read/write)

The value of **UDCAUSE** for precise exception:

- **When ucause == 1** (Instruction access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP instruction access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>PMA empty hole access</td>
</tr>
</tbody>
</table>

- **When ucause == 2** (Illegal instruction)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Please parse the utval CSR</td>
</tr>
<tr>
<td>1</td>
<td>FP disabled exception</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- **When ucause == 5** (Load access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>PMP load access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>5</td>
<td>PMA empty hole access</td>
</tr>
<tr>
<td>6</td>
<td>PMA attribute inconsistency</td>
</tr>
<tr>
<td>7</td>
<td>PMA NAMO exception</td>
</tr>
</tbody>
</table>

- When `ucause == 7` (Store access fault)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ECC/Parity error</td>
</tr>
<tr>
<td>2</td>
<td>PMP store access violation</td>
</tr>
<tr>
<td>3</td>
<td>Bus error</td>
</tr>
<tr>
<td>4</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>5</td>
<td>PMA empty hole access</td>
</tr>
<tr>
<td>6</td>
<td>PMA attribute inconsistency</td>
</tr>
<tr>
<td>7</td>
<td>PMA NAMO exception</td>
</tr>
</tbody>
</table>

### 16.12 Memory and Miscellaneous Registers

#### 16.12.1 Instruction Local Memory Base Register

**Mnemonic Name:** milmb  
**IM Requirement:** `ILM_SIZE_KB > 0`  
**Access Mode:** Machine  
**CSR Address:** 0x7c0 (non-standard read/write)

![MILMB Register Diagram](image)

This register controls instruction local memory.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEN</td>
<td>[0]</td>
<td>ILM enable control:</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>ILM is disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>ILM is enabled</td>
<td></td>
</tr>
<tr>
<td>ECCEN</td>
<td>[2:1]</td>
<td>Parity/ECC enable control:</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable parity/ECC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Generate exceptions only on uncorrectable parity/ECC errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Generate exceptions on any type of parity/ECC errors</td>
<td></td>
</tr>
<tr>
<td>RWECC</td>
<td>[3]</td>
<td>Controls diagnostic accesses of ECC codes of the ILM RAMs. When set, load/store to ILM reads/writes ECC codes to the mecc_code register. This bit can be set for injecting ECC errors to test the ECC handler.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td>IBPA</td>
<td>[63:10]</td>
<td>The base physical address of ILM. It has to be an integer multiple of the ILM size.</td>
<td>RO</td>
<td>ILM_BASE[63:10]</td>
</tr>
</tbody>
</table>
16.12.2  Data Local Memory Base Register

Mnemonic Name: mdlmb
IM Requirement: DLM_SIZE_KB > 0
Access Mode: Machine
CSR Address: 0x7c1 (non-standard read/write)

This register controls data local memory.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEN</td>
<td>[0]</td>
<td>DLM enable control:</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>DLM is disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>DLM is enabled</td>
</tr>
<tr>
<td>ECCEN</td>
<td>[2:1]</td>
<td>Parity/ECC enable control:</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Disable parity/ECC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>Generate exceptions only on uncorrectable parity/ECC errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td>Generate exceptions on any type of parity/ECC errors</td>
</tr>
<tr>
<td>RWECC</td>
<td>[3]</td>
<td>Controls diagnostic accesses of ECC codes of the DLM RAMs. When set, load/store to DLM reads/writes ECC codes to the mecc_code register. This bit can be set for injecting ECC errors to test the ECC handler.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Disable diagnostic accesses of ECC codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Enable diagnostic accesses of ECC codes</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPA</td>
<td>[63:10]</td>
<td>The base physical address of DLM. It has to be an integer multiple of the DLM size.</td>
<td>RO</td>
<td>DLM_BASE[63:10]</td>
</tr>
</tbody>
</table>
16.12.3  ECC Code Register

Mnemonic Name: mecc_code  
IM Requirement: mmSC_cfg.ECC == 1 
Access Mode: Machine  
CSR Address: 0x7c2 (non-standard read/write)

This register is used for accessing ECC array.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>[7:0]</td>
<td>This field records the ECC value on ECC error exceptions. This field is also used to read/write the ECC codes when diagnostic access of ECC codes are enabled (milmb.RWECC or mdlmb.RWECC is 1).</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>[16]</td>
<td>Correctable error. This bit is updated on parity/ECC error exceptions.</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Uncorrectable error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Correctable error</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>[17]</td>
<td>Precise error. This bit is updated on parity/ECC error exceptions.</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Imprecise error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Precise error</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
### Field Name | Bits | Description | Type | Reset
--- | --- | --- | --- | ---
RAMID | [21:18] | The ID of RAM that caused parity/ECC errors. This bit is updated on parity/ECC error exceptions. | RO | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Tag RAM of I-Cache</td>
</tr>
<tr>
<td>3</td>
<td>Data RAM of I-Cache</td>
</tr>
<tr>
<td>4</td>
<td>Tag RAM of D-Cache</td>
</tr>
<tr>
<td>5</td>
<td>Data RAM of D-Cache</td>
</tr>
<tr>
<td>6</td>
<td>Tag RAM of TLB</td>
</tr>
<tr>
<td>7</td>
<td>Data RAM of TLB</td>
</tr>
<tr>
<td>8</td>
<td>ILM</td>
</tr>
<tr>
<td>9</td>
<td>DLM</td>
</tr>
<tr>
<td>10–15</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### INSN | [22] | Indicates if the parity/ECC error is caused by instruction fetch or data access. | RO | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data access</td>
</tr>
<tr>
<td>1</td>
<td>Instruction fetch</td>
</tr>
</tbody>
</table>
### 16.12.4 NMI Vector Base Address Register

**Mnemonic Name:** mnvec  
**IM Requirement:** Required  
**Access Mode:** Machine  
**CSR Address:** 0x7e3 (non-standard read/write)

This register indicates the entry point when an NMI occurs.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNVEC</td>
<td>[63:0]</td>
<td>Base address of the NMI handler. Its value is the program starting address when NMI occurs.</td>
<td>RO</td>
<td>Pin Configured</td>
</tr>
</tbody>
</table>
16.12.5 Performance Throttling Control Register

Mnemonic Name: mpft_ctl
IM Requirement: POWERBRAKE_SUPPORT = "yes"
Access Mode: Machine
CSR Address: 0x7c5 (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_LEVEL</td>
<td>[7:4]</td>
<td>Throttling Level. The processor has the highest performance at throttling level 0 and the lowest performance at throttling level 15.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Level 0 (the highest performance)</td>
</tr>
<tr>
<td>1-14</td>
<td>Level 1-14</td>
</tr>
<tr>
<td>15</td>
<td>Level 15 (the lowest performance)</td>
</tr>
</tbody>
</table>

Field Name: FAST_INT [8]
Description: Fast interrupt response. If this field is set, mxstatus.PFT_EN will be automatically cleared when the processor enters an interrupt handler.
Type: RW
Reset: 0
### 16.12.6 Cache Control Register

**Mnemonic Name:** mcache_ctl  
**IM Requirement:** Cache optional (micm_cfg.ISZ != 0 or mdcm_cfg.DSZ != 0)  
**Access Mode:** Machine  
**CSR Address:** 0x7ca (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_EN</td>
<td>[0]</td>
<td>Controls if the instruction cache is enabled or not.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>I-Cache is disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I-Cache is enabled</td>
<td></td>
</tr>
<tr>
<td>DC_EN</td>
<td>[1]</td>
<td>Controls if the data cache is enabled or not.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>D-Cache is disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>D-Cache is enabled</td>
<td></td>
</tr>
<tr>
<td>IC_ECCEN</td>
<td>[3:2]</td>
<td>Parity/ECC error checking enable control for the instruction cache.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable parity/ECC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Generate exceptions only on uncorrectable parity/ECC errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Generate exceptions on any type of parity/ECC errors</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_ECCEN</td>
<td>[5:4]</td>
<td>Parity/ECC error checking enable control for the data cache.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable parity/ECC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Generate exceptions only on uncorrectable parity/ECC errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Generate exceptions on any type of parity/ECC errors</td>
<td></td>
</tr>
<tr>
<td>IC_RWECC</td>
<td>[6]</td>
<td>Controls diagnostic accesses of ECC codes of the instruction cache RAMs. It is set to enable CCTL operations to access the ECC codes (see Section 7.6). This bit can be set for injecting ECC errors to test the ECC handler.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td>DC_RWECC</td>
<td>[7]</td>
<td>Controls diagnostic accesses of ECC codes of the data cache RAMs. It is set to enable CCTL operations to access the ECC codes (see Section 7.6). This bit can be set for injecting ECC errors to test the ECC handler.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enable diagnostic accesses of ECC codes</td>
<td></td>
</tr>
<tr>
<td>Field Name</td>
<td>Bits</td>
<td>Description</td>
<td>Type</td>
<td>Reset</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>CCTL_SUEN</td>
<td>[8]</td>
<td>Enable bit for Superuser-mode and User-mode software to access ucctlbeginaddr and ucctlcommand CSRs.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Disable ucctlbeginaddr and ucctlcommand accesses in S/U mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable ucctlbeginaddr and ucctlcommand accesses in S/U mode</td>
</tr>
<tr>
<td>IPREF_EN</td>
<td>[9]</td>
<td>This bit controls hardware prefetch for instruction fetches to cacheable memory regions when I-Cache size is not 0.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Disable hardware prefetch on instruction fetches</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable hardware prefetch on instruction fetches</td>
</tr>
<tr>
<td>DPREF_EN</td>
<td>[10]</td>
<td>This bit controls hardware prefetch for load/store accesses to cacheable memory regions when D-Cache size is not 0.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Disable hardware prefetch on load/store memory accesses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Enable hardware prefetch on load/store memory accesses.</td>
</tr>
<tr>
<td>IC_FIRST_WORD</td>
<td>[11]</td>
<td>I-Cache miss allocation filling policy</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Cache line data is returned critical (double) word first</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Cache line data is returned the lowest address (double) word first</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_FIRST_WORD</td>
<td>[12]</td>
<td>D-Cache miss allocation filling policy</td>
<td>RO</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0           Cache line data is returned critical (double) word first</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1           Cache line data is returned the lowest address (double) word first</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC_WAROUND</td>
<td>[14:13]</td>
<td>D-Cache Write-Around threshold</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0           Disables streaming. All cacheable write misses allocate a cache line according to PMA settings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1           Stop allocating D-Cache entries regardless of PMA settings after consecutive stores to 4 cache lines.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2           Stop allocating D-Cache entries regardless of PMA settings after consecutive stores to 64 cache lines.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3           Stop allocating D-Cache entries regardless of PMA settings after consecutive stores to 128 cache lines.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC_COHEN</td>
<td>[19]</td>
<td>Enable data cache to participate in the coherence management.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0           Disable Data cache to participate in the coherence management</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1           Enable Data cache to participate in the coherence management</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_COHSTA</td>
<td>[20]</td>
<td>Indicate if data cache is participated in the coherence management</td>
<td>RO</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data cache is not participated in the coherence management</td>
</tr>
<tr>
<td>1</td>
<td>Data cache is participated in the coherence management</td>
</tr>
</tbody>
</table>
16.12.7 Machine Miscellaneous Control Register

Mnemonic Name: mmisc_ctl
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7d0 (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEC_PLIC</td>
<td>[1]</td>
<td>Selects the operation mode of PLIC:</td>
</tr>
<tr>
<td>RVCOMPM</td>
<td>[2]</td>
<td>RISC-V compatibility mode enable bit. If the compatibility mode is turned on, all Andes-specific instructions become reserved instructions.</td>
</tr>
</tbody>
</table>

**Value** | **Meaning**
--- | ---
0 | Regular mode
1 | Vector mode

Please note that both this bit and the vector mode enable bit (VECTORED) of the Feature Enable Register in NCEPLIC100 should be turned on for the vectored interrupt support to work correctly. See Section 18.4.2 for the definition of the VECTORED bit.

This bit is hardwired to 0 if the vectored PLIC feature is not supported.

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRPE</td>
<td>[3]</td>
<td>Branch prediction enable bit. This bit controls all branch prediction structures.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>MSA/UNA</td>
<td>[6]</td>
<td>This field controls whether the load/store instructions can access misaligned memory locations without generating Address Misaligned exceptions. Supported instructions: LW/LH/LHU/SW/SH/LD/LWU/SD</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Misaligned accesses generate Address Misaligned exceptions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Misaligned accesses are allowed.</td>
<td></td>
</tr>
<tr>
<td>NBLD_EN</td>
<td>[8]</td>
<td>This field controls the blocking behavior of load instructions. When this bit is clear, load instructions accessing non-device regions are blocking. When this bit is set, load instructions will not be blocking on such occasions and bus errors will no longer be reported synchronously. See Section 11.1 for details.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Load to memory regions are blocking.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Load to memory regions are non-blocking.</td>
<td></td>
</tr>
</tbody>
</table>
16.12.8 Clock Control Register

Mnemonic Name: mclk_ctl  
IM Requirement: (misa.V == 1) | (mmse_cfg.BF16CVT == 1)  
Access Mode: Machine  
CSR Address: 0x7df (non-standard read/write)

The floating point unit (FPU) is implemented with 3 levels of dynamic clock gating. This clock gating scheme is beyond that generated automatically from the synthesis tool. The clock gating functions are enabled by default. Users can disable the specific clock gating function by programming mclk_ctl register.

- The first level (level 1) is from functional modules within a functional unit. This level is not visible to the top level.
- The second level (level 2) is per functional unit. This level is visible to the top level.
- The third level (level 3) is FPU idle signal which combines all idle signals. This level is visible to the top level.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKGATE</td>
<td>[7:0]</td>
<td>One-hot clock gating levels.</td>
<td>RW</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Bit</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Level 1 clock gating in module level</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Level 2 clock gating in unit level</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Level 3 clock gating in VPU level</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7:3</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>FP</td>
<td>[10]</td>
<td>Level 1 clock gating enable for scalar floating point issue unit and queues.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>UQ</td>
<td>[11]</td>
<td>Level 1 clock gating enable for uncached queues.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>DQ</td>
<td>[12]</td>
<td>Level 1 clock gating enable for data cache load/store queues.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>AQ</td>
<td>[13]</td>
<td>Level 1 clock gating enable for ACE load/store queues.</td>
<td>RW</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR</td>
<td>[14]</td>
<td>Level 1 clock gating enable for the vector/floating-point register file.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>VI</td>
<td>[15]</td>
<td>Level 1 clock gating enable for the vector/floating-point issue queues.</td>
<td>RW</td>
<td>1</td>
</tr>
<tr>
<td>FUNIT</td>
<td>[31:16]</td>
<td>Level 2 clock gating enable for function units listed in the following table.</td>
<td>RW</td>
<td>0x1ff</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>integer arithmetic unit</td>
</tr>
<tr>
<td>17</td>
<td>integer permutation unit</td>
</tr>
<tr>
<td>18</td>
<td>integer mask unit</td>
</tr>
<tr>
<td>19</td>
<td>integer division unit</td>
</tr>
<tr>
<td>20</td>
<td>integer multiply and add unit</td>
</tr>
<tr>
<td>21</td>
<td>floating-point multiply and add unit</td>
</tr>
<tr>
<td>22</td>
<td>floating-point miscellaneous unit</td>
</tr>
<tr>
<td>23</td>
<td>floating-point division unit</td>
</tr>
<tr>
<td>24</td>
<td>load/store unit</td>
</tr>
<tr>
<td>31:25</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Note**
- When RISC-V vector extension is not supported, the following control fields are not used in FPU although they are writable by software. ⇒ mclk_ctl[10], mclk_ctl[11], mclk_ctl[13], and mclk_ctl[16:20]
16.12.9 Machine CCTL Begin Address

Mnemonic Name: mcctlbeginaddr
IM Requirement: Cache optional
Access Mode: Machine
CSR Address: 0x7cb (non-standard read/write)

This register holds the address information required by CCTL operations. It is only present when (micm_cfg.ISZ!=0 or mdcm_cfg.DSZ!=0) and (mmsc_cfg.CCTLCSR==1).

- For “VA” type of CCTL operations:
  The mcctlbeginaddr register contains the starting virtual address for CCTL operations triggered by writes to the mcctlcommand register. For CCTL lock operations, the mcctldata register will be updated with a status value (0:fail, 1:success) when the operations complete.
  After an update to the mcctlcommand register with a VA-type command, the value of this register will be incremented with the byte size of the corresponding cache line.

- For “Index” type of CCTL operations:
  The mcctlbeginaddr register contains the cache index for CCTL operations triggered by writes to the mcctlcommand register.

  - For all Index-type commands other than “IX_RDATA” and “IX_WDATA”:
    The INDEX field in this register will be incremented. If the incremented INDEX wraps around to 0 (i.e., the first way of a set), then the WAY field in this register will be incremented.

  - For “IX_RDATA” and “IX_WDATA” commands:
    The OFFSET field in this register will be incremented first to the next OFFSET value. If the incremented OFFSET field wraps around to 0 (i.e., the first double word of a cache line), then the INDEX field in this register will be incremented. If the incremented INDEX field wraps around to 0 (i.e., the first way of a set), then the WAY field in this register will be incremented.
16.12.10 Machine CCTL Command

Mnemonic Name: mcctlcommand
IM Requirement: Cache optional
Access Mode: Machine
CSR Address: 0x7cc (non-standard read/write)

Writing to this register will trigger a CCTL operation, with the type of the operation specified by the value written. Valid CCTL operations are defined in Table 58. See Section 7.6 for more information. CCTL operations are inherently not atomic, see notes below for usage limitations.

This register is only present when \((\text{micm}_\text{cfg}.\text{ISZ}!=0 \text{ or } \text{mdcm}_\text{cfg}.\text{DSZ}!=0)\) and \((\text{mmsc}_\text{cfg}.\text{CCTLCSR}==1)\).

<table>
<thead>
<tr>
<th>Value</th>
<th>Command</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>L1D_VA_INVAL</td>
<td>VA</td>
</tr>
<tr>
<td>1</td>
<td>L1D_VA_WB</td>
<td>VA</td>
</tr>
<tr>
<td>2</td>
<td>L1D_VA_WBINVAL</td>
<td>VA</td>
</tr>
<tr>
<td>3</td>
<td>L1D_VA_LOCK</td>
<td>VA</td>
</tr>
<tr>
<td>4</td>
<td>L1D_VA_UNLOCK</td>
<td>VA</td>
</tr>
<tr>
<td>6</td>
<td>L1D_WBINVAL_ALL</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>L1D_WB_ALL</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>L1I_VA_INVAL</td>
<td>VA</td>
</tr>
<tr>
<td>11</td>
<td>L1I_VA_LOCK</td>
<td>VA</td>
</tr>
<tr>
<td>12</td>
<td>L1I_VA_UNLOCK</td>
<td>VA</td>
</tr>
<tr>
<td>16</td>
<td>L1D_IX_INVAL</td>
<td>Index</td>
</tr>
<tr>
<td>17</td>
<td>L1D_IX_WB</td>
<td>Index</td>
</tr>
<tr>
<td>18</td>
<td>L1D_IX_WBINVAL</td>
<td>Index</td>
</tr>
<tr>
<td>19</td>
<td>L1D_IX_RTAG</td>
<td>Index</td>
</tr>
<tr>
<td>20</td>
<td>L1D_IX_RDATA</td>
<td>Index</td>
</tr>
<tr>
<td>21</td>
<td>L1D_IX_WTAG</td>
<td>Index</td>
</tr>
<tr>
<td>22</td>
<td>L1D_IX_WDATA</td>
<td>Index</td>
</tr>
<tr>
<td>23</td>
<td>L1D_INVAL_ALL</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td>L1I_IX_INVAL</td>
<td>Index</td>
</tr>
<tr>
<td>27</td>
<td>L1I_IX_RTAG</td>
<td>Index</td>
</tr>
<tr>
<td>28</td>
<td>L1I_IX_RDATA</td>
<td>Index</td>
</tr>
</tbody>
</table>

Continued on next page...
### Table 58: (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Command</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>0b11_101</td>
<td>L1I_IX_WTAG</td>
</tr>
<tr>
<td>30</td>
<td>0b11_110</td>
<td>L1I_IX_WDATA</td>
</tr>
</tbody>
</table>

**Note**

CCTL operations take parameters from multiple CSR registers, thus they are inherently not atomic. In addition, the CSRs share common storage across privilege levels, making them vulnerable to be overwritten across context switches. This implies that higher privilege level software should back up the content of CCTL CSR registers with interrupts disabled before using them, and restore their values afterwards if CCTL operations will be invoked in multiple privilege levels. The same is true if CCTL operations will be used both in non-interrupt codes and in the interrupt handlers.
## 16.12.11 Machine CCTL Data

**Mnemonic Name:** mcctldata  
**IM Requirement:** Cache optional  
**Access Mode:** Machine  
**CSR Address:** 0x7cd (non-standard read/write)

This register holds data required/returned by some CCTL operations. It is only present when \((\text{micm}_{\text{cfg.ISZ}}=0 \text{ or } \text{mdcm}_{\text{cfg.DSZ}}=0)\) and \((\text{mmse}_{\text{cfg.CCTLCSR}}=1)\). The complete list of CCTL operations are summarized in Table 59 and described below.

<table>
<thead>
<tr>
<th>Value of mcctldata command</th>
<th>Command</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3  0b00_011</td>
<td>L1D_VA_LOCK</td>
<td>VA</td>
</tr>
<tr>
<td>11 0b01_011</td>
<td>L1I_VA_LOCK</td>
<td>VA</td>
</tr>
<tr>
<td>19 0b10_011</td>
<td>L1D_IX_RTAG</td>
<td>Index</td>
</tr>
<tr>
<td>20 0b10_100</td>
<td>L1D_IX_RDATA</td>
<td>Index</td>
</tr>
<tr>
<td>21 0b10_101</td>
<td>L1D_IX_WTAG</td>
<td>Index</td>
</tr>
<tr>
<td>22 0b10_110</td>
<td>L1D_IX_WDATA</td>
<td>Index</td>
</tr>
<tr>
<td>27 0b11_011</td>
<td>L1I_IX_RTAG</td>
<td>Index</td>
</tr>
<tr>
<td>28 0b11_100</td>
<td>L1I_IX_RDATA</td>
<td>Index</td>
</tr>
<tr>
<td>29 0b11_101</td>
<td>L1I_IX_WTAG</td>
<td>Index</td>
</tr>
<tr>
<td>30 0b11_110</td>
<td>L1I_IX_WDATA</td>
<td>Index</td>
</tr>
</tbody>
</table>

- For CCTL lock operations: The mcctldata register will be updated with a status value (0: fail, 1:success) when the operations complete.

- For CCTL index read/write-data operations: \(\text{mcctldata}[63:0]\) holds the cache data for the operations.

- For CCTL index read/write-tag operations, mcctldata register holds the cache tag for the operations.
  - The bit positions of the fields for I-Cache CCTL index read/write-tag is as follow:
    - The TAG field does not hold every bits of \(\text{PA}[(\text{PALEN-1}):10]\) for the cache line. The cache tag RAMs do not hold all physical addresses down to bit 10. They only hold enough bits for tag look-ups. The unused lower order TAG bits will be reported as \(\text{Don't Care}\) value for tag-read operations and ignored on tag-write operations. The full PA value should be constructed using the corresponding index bits.
* I-Cache TAG RAM holds PA[(PA_LEN-1):A].
  \[ A = \min(12: (\log_2(micm_cfg.ISET) + 6)) \]
* If A is 11, bit 0 is unused. If A is 12, bit 0, 1 are unused.
* Bit XLEN-3 holds the duplicated lock bit for I-Cache to better tolerate soft-errors.
* Bit XLEN-2 holds the lock bit
* Bit XLEN-1 holds the valid bit.

The bit positions of the fields for D-Cache CCTL index read/write-tag is as follow:
* MESI field holds the cache line status.
* Bit 3 holds the lock bit.
* The TAG field holds the tag data for the cache line. \( DTW = \text{PALEN}-6 - \log_2(\text{mdcm_cfg.DSET}) \)

<table>
<thead>
<tr>
<th>Table 60: I-Cache CCTL Index Read/Write TAG Bit Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>TAG</td>
</tr>
<tr>
<td>LOCK_DUP</td>
</tr>
<tr>
<td>LOCK</td>
</tr>
<tr>
<td>VALID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 61: D-Cache CCTL Index Read/Write TAG Bit Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>MESI</td>
</tr>
<tr>
<td>LOCK</td>
</tr>
<tr>
<td>TAG</td>
</tr>
</tbody>
</table>
16.12.12 Supervisor CCTL Data

Mnemonic Name: scctldata
IM Requirement: Cache optional
Access Mode: Supervisor and above
CSR Address: 0x9cd (non-standard read/write)

This register is only present when ((micm_cfg.ISZ!=0 or mdcm_cfg.DSZ!=0) and mmsc_cfg.CCTLCSR==1 and misa[18]==1). It is an alias to the mcctldata register and it is only accessible to Supervisor-mode software when mcache_ctl.CCTL_SUEN is 1. Otherwise illegal instruction exceptions will be triggered.

Note
• S-mode software triggers CCTL operations through writing the ucctlcommand register, and associated addresses for the CCTL operations are specified in the ucctlbeginaddr register.
• Due to the sharing of storage with mcctldata, interrupt-handler/M-mode software should back up mcctldata before use; see notes in Section 16.12.10 for usage limitations.
16.12.13 User CCTL Begin Address

Mnemonic Name: ucctlbeginaddr  
IM Requirement: Cache optional  
Access Mode: User and above  
CSR Address: 0x80b (non-standard read/write)

This register is only present when ((micm_cfg.ISZ!=0 or mdcm_cfg.DSZ!=0) and mmsc_cfg.CCTLCSR==1 and misa[20]==1). It is an alias to the mcctlbeginaddr register and it is only accessible to Supervisor-mode and User-mode software when mcache_ctl.CCTL_SUEN is 1. Otherwise illegal instruction exceptions will be triggered.

Note

- Both S-mode and U-mode software triggers CCTL operations through writing the ucctlcommand register; the associated addresses for CCTL operations are specified in the ucctlbeginaddr register.
- Due to the sharing of storage with mcctlbeginaddr, interrupt-handler/privileged-mode software should back up mcctlbeginaddr before use; see notes in Section 16.12.10 for usage limitations.
16.12.14 User CCTL Command

Mnemonic Name: ucctlcommand
IM Requirement: Cache optional
Access Mode: User and above
CSR Address: 0x80c (non-standard read/write)

Writing to this register will trigger a CCTL operation, with the type of the operation specified by the value written. Valid User CCTL commands are defined in Table 62. Both S-mode and U-mode software trigger CCTL operations through this register. However, if ucctlcommand is written by U-mode software with a command whose “U-Mode Allowed” field is 0 in Table 62, an illegal instruction exception will still be generated.

This register is only present when ((micm_cfg.ISZ!=0 or mdcm_cfg.DSZ!=0) and mmsc_cfg.CCTLCSR==1 and misa[20]==1) and it is an alias to the mcctlcommand register. This register is only accessible to Supervisor-mode and User-mode software when mcache_ctl.CCTL_SUEN is 1. Otherwise illegal instruction exceptions will be triggered.

Table 62: User CCTL Command Definition

<table>
<thead>
<tr>
<th>Value of ucctlcommand</th>
<th>Command</th>
<th>Type</th>
<th>U-Mode Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0b00_000</td>
<td>L1D_VA_INVAL</td>
<td>VA</td>
<td>1</td>
</tr>
<tr>
<td>1 0b00_001</td>
<td>L1D_VA_WB</td>
<td>VA</td>
<td>1</td>
</tr>
<tr>
<td>2 0b00_010</td>
<td>L1D_VA_WBINVAL</td>
<td>VA</td>
<td>1</td>
</tr>
<tr>
<td>3 0b00_011</td>
<td>L1D_VA_LOCK</td>
<td>VA</td>
<td>0</td>
</tr>
<tr>
<td>4 0b00_100</td>
<td>L1D_VA_UNLOCK</td>
<td>VA</td>
<td>0</td>
</tr>
<tr>
<td>6 0b00_110</td>
<td>L1D_WBINVAL_ALL</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>7 0b00_111</td>
<td>L1D_WB_ALL</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>8 0b01_000</td>
<td>L1I_VA_INVAL</td>
<td>VA</td>
<td>1</td>
</tr>
<tr>
<td>11 0b01_011</td>
<td>L1I_VA_LOCK</td>
<td>VA</td>
<td>0</td>
</tr>
<tr>
<td>12 0b01_100</td>
<td>L1I_VA_UNLOCK</td>
<td>VA</td>
<td>0</td>
</tr>
<tr>
<td>16 0b10_000</td>
<td>L1D_IX_INVAL</td>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>17 0b10_001</td>
<td>L1D_IX_WB</td>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>18 0b10_010</td>
<td>L1D_IX_WBINVAL</td>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>19 0b10_011</td>
<td>L1D_IX_RTAG</td>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>20 0b10_100</td>
<td>L1D_IX_RDATA</td>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>21 0b10_101</td>
<td>L1D_IX_WTAG</td>
<td>Index</td>
<td>0</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 62: (continued)

<table>
<thead>
<tr>
<th>Value of ucctl command</th>
<th>Command</th>
<th>Type</th>
<th>U-Mode Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>0b10_110</td>
<td>L1D_IX_WDATA</td>
<td>Index</td>
</tr>
<tr>
<td>23</td>
<td>0b10_111</td>
<td>L1D_INVAL_ALL</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td>0b11_000</td>
<td>L1I_IX_INVAL</td>
<td>Index</td>
</tr>
<tr>
<td>27</td>
<td>0b11_011</td>
<td>L1I_IX_RTAG</td>
<td>Index</td>
</tr>
<tr>
<td>28</td>
<td>0b11_100</td>
<td>L1I_IX_RDATA</td>
<td>Index</td>
</tr>
<tr>
<td>29</td>
<td>0b11_101</td>
<td>L1I_IX_WTAG</td>
<td>Index</td>
</tr>
<tr>
<td>30</td>
<td>0b11_110</td>
<td>L1I_IX_WDATA</td>
<td>Index</td>
</tr>
</tbody>
</table>
16.13  Hardware Stack Protection and Recording Registers

16.13.1  Machine Hardware Stack Protection Control

Mnemonic Name:  mhsp_ctl
IM Requirement:  STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)
Access Mode:  Machine
CSR Address:  0x7C6 (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVF_EN</td>
<td>[0]</td>
<td>Enable bit for the stack overflow protection and recording mechanism. This bit will be cleared to 0 automatically by hardware when a stack protection (overflow or underflow) exception is taken.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UDF_EN</td>
<td>[1]</td>
<td>Enable bit for the stack underflow protection mechanism. This bit will be cleared to 0 automatically by hardware when a stack protection (overflow or underflow) exception is taken.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The stack overflow protection and recording mechanism are disabled.</td>
</tr>
<tr>
<td>1</td>
<td>The stack overflow protection and recording mechanism are enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The stack underflow protection is disabled.</td>
</tr>
<tr>
<td>1</td>
<td>The stack underflow protection is enabled.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Bits</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>SCHM</td>
<td>[2]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>U</td>
<td>[3]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>[4]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>[5]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Note: Stack Protection is not supported for DSP/FPU/ACE instructions.
16.13.2 Machine SP Bound Register

**Mnemonic Name:** msp_bound  
**IM Requirement:** STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)  
**Access Mode:** Machine  
**CSR Address:** 0x7c7 (non-standard read/write)

<table>
<thead>
<tr>
<th>63</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP_BOUND</td>
<td></td>
</tr>
</tbody>
</table>

When the SP overflow detection mechanism is properly selected and enabled, any updated value to the SP register (via any instruction) is compared with the msp_bound register. If the updated value to the SP register is smaller than the msp_bound register, a stack overflow exception is generated. The stack overflow exception has an exception code of 32 in the mcause register.

When the top of stack recording mechanism is properly selected and enabled, any updated value to the SP register on any instruction is compared with the msp_bound register. If the updated value to the SP register is smaller than the msp_bound register, the msp_bound register is updated with this updated value. It is an RW-type register with the all-one reset value (0xFFFFFFFF for RV32 and 0xFFFFFFFFFFFFFFFF for RV64).

**Programming Note:**

- The “CSRRW sp, msp_bound, rs” instruction updates both sp and msp_bound registers at the same time. When the stack overflow detection mechanism is enabled, using this instruction may generate unpredictable exception behavior.
16.13.3 Machine SP Base Register

Mnemonic Name: msp_base
IM Requirement: STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)
Access Mode: Machine
CSR Address: 0x7c8 (non-standard read/write)

When the SP underflow detection mechanism is properly selected and enabled, any updated value to the SP register (via any instruction) is compared with the msp_base register. If the updated value to the SP register is greater than the msp_base register, a stack underflow exception is generated. The stack underflow exception has an exception code of 33 in the mcause register.

It is an RW-type register with the all-one reset value (0xFFFFFFFF for RV32 and 0xFFFFFFFFFFFFFFFF for RV64).

Programming Note:
- The “CSRRW sp, msp_base, rs” instruction updates both sp and msp_base registers at the same time. When the stack underflow detection mechanism is enabled, using this instruction may generate unpredictable exception behavior.
16.14 CoDense Registers

16.14.1 Instruction Table Base Address Register

Mnemonic Name: uitb
IM Requirement: CODENSE_SUPPORT = "yes"
Access Mode: User and above
CSR Address: 0x800 (non-standard read/write)

This register defines the base address of the CoDense instruction table. Each entry in the table contains a 32-bit instruction, which can be looked up and executed by a CoDense instruction. The table is typically generated by the compiler for replacing 32-bit instructions with the shorter 16-bit Andes CoDense instructions, hence reducing the code size.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>[0]</td>
<td>This bit specifies if the CoDense instruction table is hardwired.</td>
<td>RO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>The instruction table is located in memory. <strong>uitb.ADDR</strong> should be initialized to point to the table before using the CoDense instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The instruction table is hardwired. Initialization of <strong>uitb.ADDR</strong> is not needed before using the CoDense instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ADDR | [63:2] | The base address of the CoDense instruction table. This field is reserved if **uitb.HW** == 1. | RW | 0 |
16.15 DSP Registers

16.15.1 Code Register

Mnemonic Name: ucode
IM Requirement: DSP_SUPPORT = "yes" (mmsc_cfg.EDSP == 1)
Access Mode: User
CSR Address: 0x801 (non-standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>OV</td>
<td>[0]</td>
<td>Overflow flag. It will be set by DSP instructions with a saturated result.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A saturated result is not generated.</td>
</tr>
<tr>
<td>1</td>
<td>A saturated result is generated.</td>
</tr>
</tbody>
</table>
16.16 Physical Memory Protection Unit Configuration & Address Registers

16.16.1 PMP Configuration Registers

Mnemonic Name: pmcfg0 and pmcfg2
IM Requirement: PMP_SUPPORT
Access Mode: Machine
CSR Address: 0x3A0 and 0x3A2 (standard read/write)

0x3A0

PMP Configuration Format (for PMP_CFG)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>[0]</td>
<td>Read access control.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>[1]</td>
<td>Write access control.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>[2]</td>
<td>Instruction execution control.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[4:3]</td>
<td>Address matching mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF: Null region.</td>
</tr>
<tr>
<td>1</td>
<td>TOR: Top of range. For PMP entry 0, it matches any address $A &lt; \text{pmpaddr}_0$. For PMP entry $i$, it matches any address $A$ such that $\text{pmpaddr}<em>i &gt; A &gt;= \text{pmpaddr}</em>{i-1}$. But the 4-byte range is not supported.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved.</td>
</tr>
<tr>
<td>3</td>
<td>NAPOT: Naturally aligned power-of-2 region. This mode makes use of the low-order bits of the associated address register to encode the size of the range. See Table 63 for range encoding from the value of a PMP address register. The minimal size of NAPOT regions must be 8 bytes.</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>[7]</td>
<td>Write lock and permission enforcement bit for Machine mode.</td>
<td>W1S*</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Machine mode writes to PMP entry registers are allowed. R/W/X permissions apply to S and U modes.</td>
</tr>
<tr>
<td>1</td>
<td>For PMP entry $i$, writes to ( \text{PMP}<em>{i}\text{CFG} ) and ( \text{PMPADDR}</em>{i} ) are ignored. Additionally, if ( \text{PMP}<em>{i}\text{CFG}.A ) is set to TOR, writes to ( \text{pmpaddr}</em>{i-1} ) are ignored as well. As for permission enforcement, R/W/X permissions apply to all modes. This bit can only be cleared to 0 with a system reset.</td>
</tr>
</tbody>
</table>

**Note**

The register type of the $L$ field is W1S because only a system reset can clear this bit.
16.16.2 PMP Address Register

Mnemonic Name: pmpaddr0–pmpaddr15
IM Requirement: PMP_SUPPORT
Access Mode: Machine
CSR Address: 0x3b0 to 0x3bf (standard read/write)

Each PMP address register encodes bits 55–2 of a 56-bit physical address, as shown in the register format. Different address matching mode also decides the way how PMP addressing and memory size.

If PMP configuration field A is set as 1, address matching mode becomes TOR (Top of range) mode. At TOR mode, PMP entry 0 matches any address of A < pmpaddr0. PMP entry i matches any address of pmpaddr(i-1) <= A < pmpaddr(i), where i ranges from 1 and 15. When i is 1, PMP entry 1 matches any address of pmpaddr0 <= A < pmpaddr1. The start address of PMP entry 1 is bit[55:2] of pmpaddr0, the end address is (bit[55:2]-1) of pmpaddr1 and the Memory size is (end address - start address + 1)*4 bytes.

If PMP configuration field A is set as 3, address matching mode becomes NAPOT (Naturally aligned power-of-2 region) mode. At NAPOT mode, not all physical address bits may be implemented. The encoding is described in Table 63. “a” is an arbitrary value representing one bit value of the physical address.

<table>
<thead>
<tr>
<th>Register Content</th>
<th>Match Size(Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaa...aaa0</td>
<td>8</td>
</tr>
<tr>
<td>aaaa...aa01</td>
<td>16</td>
</tr>
<tr>
<td>aaaa...a011</td>
<td>32</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>aa01...1111</td>
<td>$2^{XLEN}$</td>
</tr>
<tr>
<td>a011...1111</td>
<td>$2^{XLEN+1}$</td>
</tr>
<tr>
<td>0111...1111</td>
<td>$2^{XLEN+2}$</td>
</tr>
<tr>
<td>1111...1111</td>
<td>$2^{XLEN+3}$ *1</td>
</tr>
</tbody>
</table>

The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.
1. In Andes implementation, the behavior of this register content is the same as that of 0111...1111 and hence the match size is equivalent to $2^{\text{XLEN}+2}$.

The smallest PMP entry granularity is 8-bytes and pmpaddr$_i[0]$ is hardwired to zero when the mode is OFF or TOR.

Note
When PMP is used in the cacheable memory space, a PMP region must also naturally align to the size of the cache line. Otherwise, a deliberate load to a cache line partially covered by a PMP region may bring the full cache line to the Data Cache and allow CCTL operations to access the reset of the cache line that may have a different access restriction.
16.17 Physical Memory Attribute Unit Configuration & Address Registers

16.17.1 PMA Configuration Registers

Mnemonic Name: pmacfg0 and pmacfg2
IM Requirement: PPMA_SUPPORT
Access Mode: Machine
CSR Address: 0xBC0 and 0xBC2 (standard read/write)

0xBC0

<table>
<thead>
<tr>
<th>63</th>
<th>56</th>
<th>48</th>
<th>47</th>
<th>40</th>
<th>39</th>
<th>32</th>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMA7CFG</td>
<td>PMA6CFG</td>
<td>PMA5CFG</td>
<td>PMA4CFG</td>
<td>PMA3CFG</td>
<td>PMA2CFG</td>
<td>PMA1CFG</td>
<td>PMA0CFG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0xBC2

<table>
<thead>
<tr>
<th>63</th>
<th>56</th>
<th>48</th>
<th>47</th>
<th>40</th>
<th>39</th>
<th>32</th>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMA15CFG</td>
<td>PMA14CFG</td>
<td>PMA13CFG</td>
<td>PMA12CFG</td>
<td>PMA11CFG</td>
<td>PMA10CFG</td>
<td>PMA9CFG</td>
<td>PMA8CFG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PMA Configuration Format (for PMAiCFG)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>NAMO</td>
<td>MTYP</td>
<td>ETYP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETYP</td>
<td>[1:0]</td>
<td>Entry address matching mode.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF: This PMA entry is disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved.</td>
</tr>
<tr>
<td>3</td>
<td>NAPOT: Naturally aligned power-of-2 region. The granularity is 4K bytes. This mode makes use of the low-order bits of the associated address register to encode the size of the range. See Table 64 for range encoding from the value of a PMA address register.</td>
</tr>
</tbody>
</table>

This field will be 0 when it is set to 1 or 2.

Continued on next page...
Field Name | Bits | Description | Type | Reset
---|---|---|---|---
MTYP | [5:2] | Memory type attribute. This field defines the cacheability and idempotency of memory regions. In the table below, “Device” regions are non-idempotent regions and “Memory” regions are idempotent. The non-cacheable memory regions (type 2 and 3) are also referred to as uncached regions by this document. | RW | 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device, Non-bufferable</td>
</tr>
<tr>
<td>1</td>
<td>Device, bufferable</td>
</tr>
<tr>
<td>2</td>
<td>Memory, Non-cacheable, Non-bufferable</td>
</tr>
<tr>
<td>3</td>
<td>Memory, Non-cacheable, Bufferable</td>
</tr>
<tr>
<td>4</td>
<td>Reserved. Hardware converts the written value to 3</td>
</tr>
<tr>
<td>5</td>
<td>Reserved. Hardware converts the written value to 3</td>
</tr>
<tr>
<td>6</td>
<td>Reserved. Hardware converts the written value to 3</td>
</tr>
<tr>
<td>7</td>
<td>Reserved. Hardware converts the written value to 3</td>
</tr>
<tr>
<td>8</td>
<td>Memory, Write-back, No-allocate</td>
</tr>
<tr>
<td>9</td>
<td>Memory, Write-back, Read-allocate</td>
</tr>
<tr>
<td>10</td>
<td>Memory, Write-back, Write-allocate</td>
</tr>
<tr>
<td>11</td>
<td>Memory, Write-back, Read and Write-allocate</td>
</tr>
<tr>
<td>12 – 14</td>
<td>Reserved. Hardware converts the written value to 15</td>
</tr>
<tr>
<td>15</td>
<td>Empty hole, nothing exists. The instruction fetch will generate an instruction access fault. The load instruction access will generate a load access fault. The store instruction access will generate a store access fault.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Bits</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>NAMO</td>
<td>[6]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AMO instructions (including LR/SC) are supported in the region.</td>
</tr>
<tr>
<td>1</td>
<td>AMO instructions (including LR/SC) are not supported in the region.</td>
</tr>
</tbody>
</table>
16.17.2 PMA Address Register

Mnemonic Name: pmaaddr0–pmaaddr15
IM Requirement: PPMA_SUPPORT
Access Mode: Machine
CSR Address: 0xBD0 to 0xBDf (standard read/write)

Each PMA address register encodes bits (PALEN-1)–2 physical address, as shown in the register format. Not all physical address bits may be implemented. The encoding is described in Table 64. “a” in the table represents one bit address, with arbitrary values.

Table 64: AX45MP-1C NAPOT Range Encoding in PMA Address and Configuration Registers

<table>
<thead>
<tr>
<th>Register Content</th>
<th>Match Size(Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaa…aaaaaaaaaaa</td>
<td>Reserved</td>
</tr>
<tr>
<td>…</td>
<td>Reserved</td>
</tr>
<tr>
<td>aaaa…aa011111111</td>
<td>Reserved</td>
</tr>
<tr>
<td>aaaa…a0111111111</td>
<td>2^{12}</td>
</tr>
<tr>
<td>aaaa…011111111</td>
<td>2^{13}</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>aa01…11111111111</td>
<td>2^{XLEN}</td>
</tr>
<tr>
<td>a011…1111111111</td>
<td>2^{XLEN+1}</td>
</tr>
<tr>
<td>0111…1111111111</td>
<td>2^{XLEN+2}</td>
</tr>
<tr>
<td>1111…11111111111</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
16.18 Floating-Point CSRs

16.18.1 Floating-Point Accrued Exception Flags

Mnemonic Name: fflags
IM Requirement: F or D ISA extension optional
Access Mode: User
CSR Address: 0x001 (standard read/write)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>NX</td>
<td>[0]</td>
<td>Inexact exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UF</td>
<td>[1]</td>
<td>Under flow exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>OF</td>
<td>[2]</td>
<td>Over flow exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>DZ</td>
<td>[3]</td>
<td>Divide by zero flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>NV</td>
<td>[4]</td>
<td>Invalid operation flag</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

This register is an aliased portion of the fcsr register.

16.18.2 Floating-Point Rounding Mode

Mnemonic Name: frm
IM Requirement: F or D ISA extension optional
Access Mode: User
CSR Address: 0x002 (standard read/write)
### Field Name Bits Description Type Reset

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRM</td>
<td>[2:0]</td>
<td>Floating-point instruction dynamic round mode control:</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Value Meaning

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RNE: Round to nearest, ties to even.</td>
</tr>
<tr>
<td>1</td>
<td>RTZ: Round towards zero.</td>
</tr>
<tr>
<td>2</td>
<td>RDN: Round down (towards -infinity)</td>
</tr>
<tr>
<td>3</td>
<td>RUP: Round up (towards + infinity)</td>
</tr>
<tr>
<td>4</td>
<td>RMM: Round to nearest, ties to max magnitude</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

This register is an aliased portion of the \( fcsr \) register with the field position shifted to LSB.

### 16.18.3 Floating-Point Control and Status

**Mnemonic Name**: \( fcsr \)

**IM Requirement**: F or D ISA extension optional

**Access Mode**: User

**CSR Address**: 0x003 (standard read/write)

### Field Name Bits Description Type Reset

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>NX</td>
<td>[0]</td>
<td>Inexact exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>UF</td>
<td>[1]</td>
<td>Under flow exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>OF</td>
<td>[2]</td>
<td>Over flow exception flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>DZ</td>
<td>[3]</td>
<td>Divide by zero flag</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>NV</td>
<td>[4]</td>
<td>Invalid operation flag</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRM</td>
<td>[7:5]</td>
<td>Floating-point instruction dynamic round mode control:</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>RNE: Round to nearest, ties to even.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>RTZ: Round towards zero.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td>RDN: Round down (towards -infinity)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td>RUP: Round up (towards +infinity)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td>RMM: Round to nearest, ties to max magnitude</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>5</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>6</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>7</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
16.19 User Counter Related CSRs

16.19.1 Cycle Counter

Mnemonic Name: `cycle`
IM Requirement: `misa.U==1`
Access Mode: User
CSR Address: 0xc00 (standard read only)

This register is the read-only shadow of `mcycle` register. Writing of this register in any mode will cause an illegal instruction exception (mcause==2). When the `mcounteren.CY` bit is cleared, attempts to read this register in User mode will cause an illegal instruction exception (mcause==2).

When `mmsc_cfg.PMNDS==1`, writing of this register in any mode is controlled by the `mcounterwen` CSR. When the corresponding counter bit in `mcounterwen` is 0, writing of this register in any mode will cause an illegal instruction exception (mcause==2). When the corresponding counter bit in `mcounterwen` is 1, writing of this register in M-mode or S-mode is allowed for an M/S/U system.
16.19.2 User Time Register

Mnemonic Name: time
IM Requirement: misa.U == 1
Access Mode: User
CSR Address: 0xc01 (software emulation)

This is the register for RDTIME instruction. An implementation can trap the RDTIME instruction and use a machine mode trap handler to get the timer value to emulate the correct behavior of a RDTIME instruction.
16.19.3 Instruction-Retired Counter

**Mnemonic Name:** instret  
**IM Requirement:** misa.U == 1  
**Access Mode:** User  
**CSR Address:** 0xc02 (standard read only)

This register is the read-only shadow of mcycle register. Writing of this register in any mode will cause an illegal instruction exception (mcause==2). When the mcounteren.CY bit is cleared, attempts to read this register in User mode will cause an illegal instruction exception (mcause==2).

When mmSc_cfg.PMNDs==1, writing of this register in any mode is controlled by the mcounterwen CSR. When the corresponding counter bit in mcounterwen is 0, writing of this register in any mode will cause an illegal instruction exception (mcause==2). When the corresponding counter bit in mcounterwen is 1, writing of this register in M-mode or S-mode is allowed for an M/S/U system.
16.19.4 Performance Monitoring Counter

**Mnemonic Name:** hpmcounter3–hpmcounter6  
**IM Requirement:** misa.U==1  
**Access Mode:** User  
**CSR Address:** 0xc03 to 0xc06 (standard read only)

These registers are the read-only shadow of mhpmcounter3–6 registers. Writing of these registers in any mode will cause an illegal instruction exception (mcause==2). When the mcounteren. HPM3–6 bits are cleared, attempts to read these registers in User mode will cause an illegal instruction exception (mcause==2).

When mmsc_cfg.PMNDS==1, writing of this register in any mode is controlled by the mcounterwen CSR. When the corresponding counter bit in mcounterwen is 0, writing of this register in any mode will cause an illegal instruction exception (mcause==2). When the corresponding counter bit in mcounterwen is 1, writing of this register in M-mode or S-mode is allowed for an M/S/U system.
17 Instruction Throughput and Latency

This chapter lists instruction throughput and latency. The instruction throughput is the number of cycles before executing the next independent instruction of the same kind. The instruction latency is the number of cycles before executing the next instruction with read-after-write dependency.

17.1 ALU Instructions

The latency and the throughput of ALU instructions are both 1 cycle. ALU instructions include:

- Add/Sub: ADD, SUB, ADDI, ADDW, SUBW, ADDIW
- Shift: SLL, SRL, SRA, SLLI, SRLI, SRAI, SLLW, SRLW, SRAW, SLLIW, SRLIW, SRAIW
- Logical: AND, OR, XOR, ANDI, ORI, XORI
- Compare: SLT, SLTU, SLTI, SLTIU
- LUI and AUIPC
- Load effective address instructions
- ADDIGP
- String processing: FFB, FFZMISM, FFMISM, FLMISM
- Bit field operation: BFOS, BFOZ

17.2 Dual-Issue Capability

This section is only for CPU revision 10.0.0 and later. For earlier revisions, see earlier documents.

17.2.1 Dual-Issue Capability of Integer Instructions

AX45MP-1C can simultaneously execute two independent instructions. Table 65 shows dual-issue capability of integer instruction pairs.

<table>
<thead>
<tr>
<th>The First Instruction</th>
<th>The Second Instruction</th>
<th>Dual-Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>Yes</td>
</tr>
<tr>
<td>ALU</td>
<td>Branch/Jump</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Continued on next page...
Table 65: (continued)

<table>
<thead>
<tr>
<th>The First Instruction</th>
<th>The Second Instruction</th>
<th>Dual-Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Load/Store</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (3R/4R/2W)</td>
<td>No</td>
</tr>
<tr>
<td>Branch/Jump</td>
<td>ALU</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Branch/Jump</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (3R/4R/2W)</td>
<td>No</td>
</tr>
<tr>
<td>Load/Store</td>
<td>ALU</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Branch/Jump</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (3R/4R/2W)</td>
<td>No</td>
</tr>
<tr>
<td>MUL/DIV</td>
<td>ALU</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Branch/Jump</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (3R/4R/2W)</td>
<td>No</td>
</tr>
<tr>
<td>DSP (2R/1W)</td>
<td>ALU</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Branch/Jump</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>DSP (3R/4R/2W)</td>
<td>No</td>
</tr>
<tr>
<td>DSP (3R/4R/2W)</td>
<td>ALU</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Branch/Jump</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>MUL/DIV</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>DSP (2R/1W)</td>
<td>No</td>
</tr>
</tbody>
</table>

Continued on next page…
Table 65: (continued)

<table>
<thead>
<tr>
<th>The First Instruction</th>
<th>The Second Instruction</th>
<th>Dual-Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP (3R/4R/2W)</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

Note

- A DSP (2R/1W) instruction reads 2 integer registers, and writes 1 register.
- A DSP (3R/4R/2W) instruction reads 3/4 integer registers, or writes 2 registers.

17.3 Throughput and Latency for Aligned Load Instructions

The throughput and latency of load instructions are summarized in the following table.

Table 66: Load Instruction Throughput and Latency

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Throughput (Cycles/Instruction)</th>
<th>Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load word/dword from DLM/D-Cache</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Load byte/halfword from DLM/D-Cache</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Load word/dword from Device/Non-cacheable memory</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Load byte/halfword from Device/Non-cacheable memory</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Load word/dword from ILM</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Load byte/halfword from ILM</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Note

- The latency assumes that the dependent instruction is executed in late ALU.
- The latency assumes that non-blocking load is disabled.
- The latency assumes that load hits cache.
- The 0-cycle latency assumes that the dependent instruction and the load instruction are dual-issued.
- The latency assumes that the CORE_CLK and BUS_CLK clock ratio is 1:1.
- The latency assumes that the latency of memory access is 1 cycle.

17.4 Throughput and Latency for Misaligned Load Instructions

For misaligned data access, it will incur extra overhead in latency and throughput. Please refer to the following table for the details.
Table 67: Misaligned Load Throughput and Latency for CPU Revision 5.0.0

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Misaligned Address</th>
<th>Throughput (Cycles/Instruction)</th>
<th>Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load halfword from DLM/D-Cache</td>
<td>addr[1:0] = 1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>addr[1:0] = 3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Load word from DLM/D-Cache</td>
<td>addr[1:0] != 0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Load dword from DLM/D-Cache</td>
<td>addr[2:0] != 0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Load halfword from Non-cacheable memory</td>
<td>addr[1:0] = 1</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>addr[1:0] = 3</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>Load word from Non-cacheable memory</td>
<td>addr[1:0] != 0</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>Load dword from Non-cacheable memory</td>
<td>addr[2:0] != 0</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>Load halfword from ILM</td>
<td>addr[1:0] = 1</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>addr[1:0] = 3</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>Load word from ILM</td>
<td>addr[1:0] != 0</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>Load dword from ILM</td>
<td>addr[2:0] != 0</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

Note

- Load byte instruction is always aligned. (i.e., no extra overhead)
- Misaligned access to device memory is not supported.
- For misaligned load, the dependent instruction cannot be executed in parallel. Instead, the dependent instruction will be replayed.
- The latency assumes that dependent instruction and the misaligned load instruction are not dual-issued.
- The latency assumes that the dependent instruction is executed in late ALU.
- The latency assumes that non-blocking load is disabled.
- The latency assumes that load hits cache.
- The latency assumes that the CORE_CLK and BUS_CLK clock ratio is 1:1.
- The latency assumes that the latency of memory access is 1 cycle.
- The latency assumes that the data width of the processor is configured to 128 bits.

17.5 Divide and Remainder Instructions

The divide and remainder instructions are implemented using the non-restoring division algorithm with early termination detection.
Table 68: Divide and Remainder Instruction Throughput and Latency

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Throughput (Cycles/Instruction)</th>
<th>Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVU, REMU</td>
<td>6–69</td>
<td>6–69</td>
</tr>
<tr>
<td>DIV, REM</td>
<td>6–69</td>
<td>6–69</td>
</tr>
</tbody>
</table>

Note

- The latency of 0x8000000000000001 / 0x1 is 69 cycles.
- The latency of 0x0000000000000000 / 0x1 is 6 cycles.

17.6 Branch and Jump Instruction

The branch and jump instruction throughput is 1 cycle/instruction. Branch mis-prediction penalty is 5 cycles when the branch is resolved in EX stage. Branch mis-prediction penalty is 7 cycles when the branch is resolved in LX stage.

17.7 EXEC.IT Instruction

When an EXEC.IT instruction is decoded, the pipeline takes extra 4 cycles to fetch the instruction from CoDense instruction table. The 4-cycle penalty might be hidden if execution pipeline is stalled by earlier instructions.

17.8 CSR Instruction

17.8.1 Latency Type

Accesses to CSR with CSRRW/S/C instructions may introduce extra latencies, depending on the latency type of the CSR and the operation. The following attributes are used to describe the latency type that CSR may possess.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>Wait for the previous operation to complete in all cases. Accessing a CSR with this attribute may introduce extra latency to the CSR instruction it self.</td>
</tr>
<tr>
<td>WH</td>
<td>Wait for the previous operation to complete only to avoid hazards in the pipeline. Accessing a CSR with this attribute may introduce extra latency to the CSR instruction it self.</td>
</tr>
</tbody>
</table>

Continued on next page...
The following table summarizes the latency due to CA or WH with respect to CSR operations.

<table>
<thead>
<tr>
<th>CSR Operation</th>
<th>CSR With WA</th>
<th>CSR With WH (w/ Hazard)</th>
<th>CSR with WH (w/o Hazard)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSR read-write</td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>0 cycle</td>
</tr>
<tr>
<td>CSR read-only</td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>0 cycle</td>
</tr>
<tr>
<td>CSR write-only</td>
<td>0 cycle</td>
<td>0 cycle</td>
<td>0 cycle</td>
</tr>
</tbody>
</table>

The following table summarizes the latency due to RF with respect to CSR operations.

<table>
<thead>
<tr>
<th>CSR Operation</th>
<th>CSR With RF</th>
<th>CSR Without RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSR read-write</td>
<td>8 cycles</td>
<td>0 cycle</td>
</tr>
<tr>
<td>CSR read-only</td>
<td>0 cycle</td>
<td>0 cycle</td>
</tr>
<tr>
<td>CSR write-only</td>
<td>8 cycles</td>
<td>0 cycle</td>
</tr>
</tbody>
</table>

**Note**
- Assuming the instruction is re-fetched from ILM. If re-fetching requires accesses to the bus, the wait cycles can be longer than 8.

### 17.8.2 List of CSRs with Latency Type WH

<table>
<thead>
<tr>
<th>mvendorid</th>
<th>marchid</th>
<th>mimpid</th>
<th>mhartid</th>
</tr>
</thead>
<tbody>
<tr>
<td>mstatus</td>
<td>misa</td>
<td>medele *</td>
<td>midele</td>
</tr>
<tr>
<td>mie</td>
<td>mtvec</td>
<td>mip</td>
<td>mxstatus</td>
</tr>
<tr>
<td>mslideleg</td>
<td>sstatus</td>
<td>sdele *</td>
<td>sidele</td>
</tr>
<tr>
<td>sie</td>
<td>stvec</td>
<td>sip</td>
<td>slie</td>
</tr>
<tr>
<td>slip</td>
<td>ustatus</td>
<td>uie</td>
<td>utvec</td>
</tr>
<tr>
<td>uscratch</td>
<td>uepc</td>
<td>ucause</td>
<td>utval</td>
</tr>
<tr>
<td>uip</td>
<td>mscratch</td>
<td>mepc</td>
<td>mcause</td>
</tr>
<tr>
<td>mtval</td>
<td>mdcause</td>
<td>sscratch</td>
<td>sepc</td>
</tr>
</tbody>
</table>
### 17.8.3 List of CSRs with Latency Type WA

WA and WH are mutually exclusive, i.e., all CSRs that are not WH are WA.

### 17.8.4 List of CSRs with Latency Type RF

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvvendorid</td>
<td>marchid</td>
<td>mimpid</td>
<td>mhartid</td>
</tr>
<tr>
<td>mstatus</td>
<td>misa</td>
<td>medeleg</td>
<td>mideleg</td>
</tr>
<tr>
<td>mie</td>
<td>mtvec</td>
<td>mip</td>
<td>mxstatus</td>
</tr>
<tr>
<td>mslideleg</td>
<td>mcounteren</td>
<td>mhpmevent3</td>
<td>mhpmevent4</td>
</tr>
<tr>
<td>mhpmevent5</td>
<td>mhpmevent6</td>
<td>mcounterwen</td>
<td>mcounterinten</td>
</tr>
<tr>
<td>mcountermask_m</td>
<td>mcountermask_s</td>
<td>mcountermask_u</td>
<td>mcounterovf</td>
</tr>
<tr>
<td>micm_cfg</td>
<td>mdcm_cfg</td>
<td>mmsc_cfg</td>
<td>tdata1</td>
</tr>
<tr>
<td>tdata2</td>
<td>tdata3</td>
<td>tinfo</td>
<td>tcontrol</td>
</tr>
<tr>
<td>mcontext</td>
<td>scontext</td>
<td>mcontrol</td>
<td>icount</td>
</tr>
<tr>
<td>itrigger</td>
<td>etrigger</td>
<td>textra</td>
<td>dcsr</td>
</tr>
<tr>
<td>dexc2dbg</td>
<td>sstatus</td>
<td>sedeleg</td>
<td>sideleg</td>
</tr>
<tr>
<td>sie</td>
<td>stvec</td>
<td>scounteren</td>
<td>sip</td>
</tr>
<tr>
<td>slie</td>
<td>slip</td>
<td>satp</td>
<td>scountermask_m</td>
</tr>
<tr>
<td>scountermask_s</td>
<td>scountermask_u</td>
<td>scounterinten</td>
<td>scounterovf</td>
</tr>
<tr>
<td>shpmevent3</td>
<td>shpmevent4</td>
<td>shpmevent5</td>
<td>shpmevent6</td>
</tr>
<tr>
<td>ustatus</td>
<td>uie</td>
<td>utvec</td>
<td>uscratch</td>
</tr>
<tr>
<td>uepc</td>
<td>ucause</td>
<td>utval</td>
<td>uip</td>
</tr>
<tr>
<td>milmb</td>
<td>mdlmb</td>
<td>mecc_code</td>
<td>mnvec</td>
</tr>
<tr>
<td>mpft_ctl</td>
<td>mcache_ctl</td>
<td>mcctlcommand</td>
<td>scctlldata</td>
</tr>
<tr>
<td>ucctlbeginaddr</td>
<td>ucctlcommand</td>
<td>mmisc_ctl</td>
<td>mhsp_ctl</td>
</tr>
<tr>
<td>msp_bound</td>
<td>msp_base</td>
<td>uitb</td>
<td>ucode</td>
</tr>
<tr>
<td>pmpcfg0</td>
<td>pmpcfg2</td>
<td>pmpaddr0</td>
<td>pmpaddr1</td>
</tr>
<tr>
<td>pmpaddr2</td>
<td>pmpaddr3</td>
<td>pmpaddr4</td>
<td>pmpaddr5</td>
</tr>
<tr>
<td>pmpaddr6</td>
<td>pmpaddr7</td>
<td>pmpaddr8</td>
<td>pmpaddr9</td>
</tr>
<tr>
<td>pmpaddr10</td>
<td>pmpaddr11</td>
<td>pmpaddr12</td>
<td>pmpaddr13</td>
</tr>
<tr>
<td>pmpaddr14</td>
<td>pmpaddr15</td>
<td>pmacfg0</td>
<td>pmacfg2</td>
</tr>
<tr>
<td>pmaaddr0</td>
<td>pmaaddr1</td>
<td>pmaaddr2</td>
<td>pmaaddr3</td>
</tr>
<tr>
<td>pmaaddr4</td>
<td>pmaaddr5</td>
<td>pmaaddr6</td>
<td>pmaaddr7</td>
</tr>
<tr>
<td>pmaaddr8</td>
<td>pmaaddr9</td>
<td>pmaaddr10</td>
<td>pmaaddr11</td>
</tr>
<tr>
<td>pmaaddr12</td>
<td>pmaaddr13</td>
<td>pmaaddr14</td>
<td>pmaaddr15</td>
</tr>
</tbody>
</table>
17.9 Trap Return Instruction

The trap return instruction flushes the entire pipeline, and the penalty is 8 cycles.

17.10 FENCE Instruction

FENCE instruction flushes the entire pipeline and waits for outstanding memory access. The penalty is 13 cycles if there is no outstanding memory access.

17.11 Scalar Floating-Point Instructions for CPU Revision 10.0.0 and Later

The instruction latencies of floating-point instruction groups are shown in the following table.

Table 69: Scalar Floating-Point Instruction Throughput and Latency for CPU Revision 10.0.0 and Later Revisions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Throughput (Cycles/Instruction)</th>
<th>Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD.x, FSUB.x, FMUL.x, FMADD.x, FMSUB.x, FNMADD.x, FNMSUB.x</td>
<td>1</td>
<td>3/4 (Note)</td>
</tr>
<tr>
<td>FDIV.D, FSQRT.D</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>FDIV.S, FSQRT.S</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>FDIV.H, FSQRT.H</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>FLH, FLW, FLD</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>FSGNJ.x, FSGNJN.x, FSGNJX.x, FMIN.x, FMAX.x</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FCLASS.x, FEQ.x, FLT.x, FLE.x</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>FCVT.D.x, FCVT.S.x, FCVT.H.x, FCVT.BF16.x</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FCVT.L.x, FCVT.LU.x, FCVT.W.x, FCVT.WU.x</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>FMV.D.X, FMV.W.X, FMV.H.X</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FMV.X.D, FMV.X.W, FMV.X.H</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Note
- The latency is 4 for floating-point add, subtract, multiplier and MAC operations when double precision FPU is configured.
- The latency is 3 for floating-point add, subtract, multiplier and MAC operations when single precision FPU is configured.
17.12 DSP Instructions

The instruction latencies of DSP instruction groups are shown in the following tables.

DSP instructions include:

- Non-MUL Arithmetic Operation (8/16/32/64-bit): ADD8, SUB16, CLZ32, RSUB64, . . .
- MUL8*8: KHM8, . . .
- MUL8*8 with 32-bit ADD/SUB: SMAQA, . . .
- MUL16*16: KHM16, . . .
- MUL16*16 with 32-bit ADD/SUB: KMDA, . . .
- MUL16*16 with 64-bit ADD/SUB: SMAL, . . .
- MUL32*32: SMMUL, . . .
- MUL32*32 with 32-bit ADD/SUB: KMMAC, . . .
- MUL32*32 with 64-bit ADD/SUB: KMAR64, . . .
- MUL32*16: SMMWB, . . .
- MUL32*16 with 32-bit ADD/SUB: KMMAWB, . . .

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Throughput (Cycles/Instruction)</th>
<th>Latency (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-MUL Arithmetic Operation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MUL8*8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MUL8*8 with 32-bit ADD/SUB</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL16*16</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL16*16 with 32-bit ADD/SUB</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL16*16 with 64-bit ADD/SUB</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MUL32*32</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL32*32 with 32-bit ADD/SUB</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL32*32 with 64-bit ADD/SUB</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MUL32*16</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL32*16 with 32-bit ADD/SUB</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
18 Platform-Level Interrupt Controller (PLIC)

18.1 Introduction

Andes Platform-Level Interrupt Controller (NCEPLIC100) prioritizes and distributes global interrupts. It is compatible with RISC-V PLIC with the following features:

- Configurable interrupt trigger types
- Software-programmable interrupt generation
- Preemptive priority interrupt extension
- Vectored interrupt extension

See Section 18.2 for information regarding the Andes preemptive priority interrupt extension and Section 18.3 for information regarding the Andes vectored PLIC extension.

The block diagram of NCEPLIC100 is shown in Figure 3. Interrupt sources are connected to the interrupt gateway of the PLIC controller, at which they are converted to interrupt requests. Interrupt requests are prioritized and routed to interrupt targets (e.g., AndesCore processor cores) according to interrupt settings. Interrupt settings include enable bits, priorities, and priority thresholds, and these settings are programmable through the bus interface. Note that interrupt targets should not modify enable bits, priorities and priority thresholds if there are any un-serviced interrupts.

When a target takes the external interrupt, it should send an interrupt claim request (bus read request) to retrieve the interrupt ID, upon which the corresponding interrupt pending status bit will be cleared.

The interrupt gateway stops processing newer interrupt requests from its interrupt sources once it reports an interrupt request. When the target has serviced the interrupt, it should send the interrupt completion message (bus write request) to NCEPLIC100 such that the interrupt gateway resumes processing newer interrupt requests.

The interrupt pending bit array of the PLIC registers provides a summary of all interrupt sources status. In addition, it is also writable for setting software-programmed interrupts for the corresponding interrupt sources. See Section 18.4.4 for more information.
18.2 Support for Preemptive Priority Interrupt

NCEPLIC100 implements the Andes preemptive priority interrupt extension which enables faster responses for high-priority interrupts. This feature is enabled by setting the `PREEMPT` field (bit 0) of the Feature Enable Register (offset: 0x0000) to 1.

With this extension, if a high-priority interrupt arrives and the global interrupt is enabled (i.e., `mstatus.MIE` or `mstatus.SIE` are 1), the processor will stop servicing the current low-priority interrupt and begin servicing this new high-priority interrupt. The handling of the suspended lower-priority interrupts will resume only after the handling of the higher-priority interrupt ends. Interrupts of same or lower priorities will not cause preemption to take effect and interfere the handling of the current interrupt. They have to wait until the handling of the current interrupt finishes.

To support this feature, the PLIC core is enhanced with a preempted priority stack for each target. The stack saves and restores priorities of the nested/preempted interrupts for the target it is associated. The operation of the preempted stack is implicitly performed through two regular PLIC operations (Interrupt Claim and Interrupt Completion). See the next two subsections for more information.
18.2.1 Interrupt Claims with Preemptive Priority

When a target sends an interrupt claim message to the PLIC core, the PLIC core will atomically
determine the ID of the highest-priority pending interrupt for the target and then deassert the corre-
sponding source’s IP bit. The PLIC core will then return the ID to the target.

At the same time, the priority number in the target’s Priority Threshold Register will be saved to a
preempted priority stack for that target and the new priority number of the claimed interrupt will be
written to the Priority Threshold Register.

18.2.2 Interrupt Completion with Preemptive Priority

When a target sends an interrupt completion message to the PLIC core, in addition to forwarding the
completion message to the associated gateway, the PLIC core will restore the highest priority number
in the preempted priority stack back to the Priority Threshold Register of the target.

Note that out-of-order completion of interrupts is not allowed when this feature is turned on — the
latest claimed interrupt should be completed first.

18.2.3 Programming Sequence to Allow Preemption of Interrupts

Turning on the global interrupt enable flag (\texttt{mstatus.MIE} or \texttt{mstatus.SIE}) is all it takes to allow
the current interrupt handler to be preempted by higher priority interrupts. However, as the pre-
emptive priority stack operations do not allow out-of-order completion, some care should be taken to
make sure that the claim and completion operations are nested properly.

For the non-vectored mode single-entry interrupt handler, the global interrupt enable flag could be
turned on after the processor context are saved and Interrupt Claim is performed to allow preemption
of the current interrupt handler. At the end of interrupt handler, an Interrupt Completion message is
performed to signal that the handler has processed the interrupt and PLIC may deliver the next inter-
rupt from the same interrupt source again. As both claim and completion messages are done through
load/store instructions to device regions, they should automatically be ordered correctly. Compared
with the vectored mode interrupt handler two paragraphs below, the global interrupt flag does not
need to be disabled and no \texttt{FENCE} needs to be inserted after sending the completion message.

In summary, below is the suggested sequence for a non-vector mode interrupt handler for supporting
preemptive priority interrupts:

1. Save registers/CSRs to stack
2. Sends Interrupt Claim message to PLIC (device-load)
3. Enable global interrupt (\texttt{mstatus.MIE/SIE})
4. Handle the expected interrupt
5. Sends *Interrupt Completion* message to PLIC (device-store)
6. Restore registers/CSRs
7. Return from interrupt

For vector mode interrupt handlers (see the next section), *Interrupt Claim* is implicit when the external interrupt is taken. The global interrupt enable flag could be turned on as long as the processor context are saved to allow preemption of the current interrupt handler. However, the global interrupt flag should be turned off before *Interrupt Completion* operations are performed, since the processor will trigger the next implicit *Interrupt Claim* operation as soon as the global interrupt enable flag is turned on and cause races between *Interrupt Claim* and *Interrupt Completion*. Additionally, a `FENCE io,io` operation should be inserted after the *Interrupt Completion* operation to make sure that the completion message reaches PLIC before the interrupt handler returns, which turns on the interrupt enable flag again and cause the next *Interrupt Claim* to be performed.

In summary, below is the suggested sequence for a vector mode interrupt handler for supporting preemptive priority interrupts:

1. Save registers/CSRs to stack
2. Enable global interrupt (`mstatus.MIE/SIE`)
3. Handle the expected interrupt
4. Disable global interrupt (`mstatus.MIE/SIE`)
5. Sends *Interrupt Completion* message to PLIC (device-store)
6. Restore registers/CSRs
7. Use a `FENCE io,io` instruction to ensure that the completion message has reached PLIC.
8. Return from interrupt

### 18.3 Vectored Interrupts

NCEPLIC100 enhances the RISC-V PLIC functionality with the vector mode extension to allow the interrupt target to receive the interrupt source ID without going through the target claim request protocol. This feature can shorten the latency of interrupt handling by enabling the interrupt target to run the corresponding interrupt handler directly upon accepting the external interrupt. It is enabled by setting the `VECTORED` field of the Feature Enable Register (offset: 0x0000) to 1.
18.4 PLIC Registers

18.4.1 Summary of Registers

NCEPLIC100 registers are accessed through bus transfers, and the summary of registers is shown in Table 71.

Please note that NCEPLIC100 supports only 32-bit transfers. Behaviors of 8-bit and 16-bit transfers are UNDEFINED, and the transfers might be ignored or result in error responses.

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
<td>End</td>
<td></td>
</tr>
<tr>
<td>0x000000</td>
<td>0x000003</td>
<td>Section 18.4.2</td>
</tr>
<tr>
<td>0x000004</td>
<td>0x000007</td>
<td>Section 18.4.3</td>
</tr>
<tr>
<td>0x000008</td>
<td>0x00000B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x000FFC</td>
<td>0x000FFF</td>
<td></td>
</tr>
<tr>
<td>0x001000</td>
<td>0x00107F</td>
<td>Section 18.4.4</td>
</tr>
<tr>
<td>0x001080</td>
<td>0x0010FF</td>
<td>Section 18.4.5</td>
</tr>
<tr>
<td>0x001100</td>
<td>0x001103</td>
<td>Section 18.4.6</td>
</tr>
<tr>
<td>0x001104</td>
<td>0x001107</td>
<td>Section 18.4.7</td>
</tr>
<tr>
<td>0x002000</td>
<td>0x00207F</td>
<td>Section 18.4.8</td>
</tr>
<tr>
<td>0x002080</td>
<td>0x0020FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x002780</td>
<td>0x0027FF</td>
<td></td>
</tr>
<tr>
<td>0x200000</td>
<td>0x200003</td>
<td>Section 18.4.9</td>
</tr>
<tr>
<td>0x200004</td>
<td>0x200007</td>
<td>Section 18.4.10</td>
</tr>
<tr>
<td>0x200400</td>
<td>0x20041F</td>
<td>Section 18.4.11</td>
</tr>
<tr>
<td>0x201000</td>
<td>0x20141F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x20F000</td>
<td>0x20F41F</td>
<td></td>
</tr>
</tbody>
</table>
18.4.2 Feature Enable Register

Offset: 0x0

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREEMPT</td>
<td>[0]</td>
<td>Preemptive priority interrupt enable</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>VECTORED</td>
<td>[1]</td>
<td>Vector mode enable</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

Please note that both this bit and the `mmisc_ctl.VEC_PLIC` bit of the processor should be turned on for the vectored interrupt support to work correctly. See Section 16.12.7 for the definition of the VEC_PLIC bit.

18.4.3 Interrupt Source Priority

Offset: n*4

This register determines the priority for interrupt source \( n \) (\( 1 \leq n \leq 1023 \)).
### Field Name Bits Description Type Reset

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY</td>
<td>[31:0]</td>
<td>Interrupt source priority. The valid range of this field is determined by the MAX_PRIORITY field of the Version &amp; Maximum Priority Configuration Register.</td>
<td>RW</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never interrupt.</td>
</tr>
<tr>
<td>1–255</td>
<td>Interrupt source priority. The larger the value, the higher the priority.</td>
</tr>
</tbody>
</table>

18.4.4 Interrupt Pending

**Offset**: 0x1000 to 0x107F

These registers provide the interrupt pending status of interrupt sources, and a way for software to trigger an interrupt without relying on external devices. Every interrupt source occupies 1 bit. There are a total of 32 registers, each 32-bit wide, for 1023 interrupt sources. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

When these registers are read, the interrupt pending status of interrupt sources are returned. The pending bits could be set by writing a bit mask that specifies the bit positions to be set, and this action would result in software-programmed interrupts of the corresponding interrupt sources. The pending bits could only be cleared through the Interrupt Claim requests.

The location of the interrupt pending bit for interrupt source \( n \) (\( 1 \leq n \leq 1023 \)) can be determined by the following equations:

- Word offset address: \( 0x1000 + 4 \times \text{floor}(n/32) \)
- Bit Position: \( n \mod 32 \)

18.4.5 Interrupt Trigger Type

**Offset**: 0x1080 to 0x10FF

These registers are read-only and indicate the configured interrupt trigger type of interrupt sources. Every interrupt source occupies 1 bit. There are a total of 32 registers, each 32-bit wide, for 1023 interrupt sources. The location of the interrupt trigger type bit for interrupt source \( n \) (\( 1 \leq n \leq 1023 \)) can be determined by the following equations:
• Word offset address: $0x1080 + 4 \times \text{floor}(n/32)$
• Bit Position: $n$ modulo 32

The meaning of each bit is shown in Table 72. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Level-triggered interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Edge-triggered interrupt</td>
</tr>
</tbody>
</table>

Table 72: Meaning of Trigger Type

18.4.6 Number of Interrupt and Target Configuration Register

Offset: 0x1100

This register indicates the number of supported interrupt sources and supported targets.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_INTERRUPT</td>
<td>[15:0]</td>
<td>The number of supported interrupt sources</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td>NUM_TARGET</td>
<td>[31:16]</td>
<td>The number of supported targets</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>

18.4.7 Version & Maximum Priority Configuration Register

Offset: 0x1104

This register indicates the version and the maximum priority of PLIC implementation.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>[15:0]</td>
<td>The version of the PLIC design</td>
<td>RO</td>
<td>IM</td>
</tr>
<tr>
<td>MAX_PRIORITY</td>
<td>[31:16]</td>
<td>The maximum priority supported</td>
<td>RO</td>
<td>IM</td>
</tr>
</tbody>
</table>
18.4.8 Interrupt Enable Bits for Target $m$

Offset: (0x2000 + $m \times 128$) to (0x207F + $m \times 128$)

These registers control the routing of interrupt source $n$ to target $m$ ($1 \leq n \leq 1023$ and $m \geq 0$). Each bit controls one interrupt source. For each target, there are a total of 32 word-sized registers for controlling 1023 interrupt sources to that target. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

The location of the interrupt enable bit for interrupt source $n$ to target $m$ can be determined by the following equations:

- Word offset address: $0x2000 + 128 \times m + 4 \times \lfloor n/32 \rfloor$
- Bit position: $n$ modulo 32

The following pseudo code demonstrates how to enable interrupt $n$ for target $m$:

```c
intptr_t reg_offset = 0x2000 + 128 * m + 4 * (n/32);
int bit_position = n % 32;

volatile uint32_t *reg_pointer = (volatile uint32_t *)(PLIC_BASE+reg_offset);
*reg_pointer = *reg_pointer | (1 << bit_position);
```

18.4.9 Priority Threshold for Target $m$

Offset: 0x200000 + 4096*$m$

Each interrupt target $m$ ($m \geq 0$) is associated with one Priority Threshold Register. Only active interrupts with priorities strictly greater than the threshold will cause an interrupt notification to be sent to the target.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLD</td>
<td>[31:0]</td>
<td>Interrupt priority threshold. The valid range of this field is determined by the MAX_PRIORITY field of the Version &amp; Maximum Priority Configuration Register.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>
18.4.10  Claim and Complete Register for Target $m$

**Offset:** 0x200004 + 4096 * $m$

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT_ID</td>
<td>[9:0]</td>
<td>On reads, indicating the interrupt source that has being claimed. On writes, indicating the interrupt source that has been handled (completed).</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

There is one Claim and Complete Register for each interrupt target $m$ ($m \geq 0$). Reading this register claims an interrupt source and returns the ID of that interrupt source.

The interrupt gateway stops processing newer interrupt requests from its interrupt sources until the earlier interrupt request completes. Writing this register with an interrupt ID serves as the interrupt completion message acknowledging to PLIC that the handling of the claimed interrupt has been serviced in target $m$ and the associated interrupt gateway may resume processing newer interrupt requests.

The interrupt gateway only resumes processing of newer interrupt requests if the enable bit of the interrupt source for target $m$ is set. If the enable bit is not set, the interrupt completion message will be ignored.

Generally there are no limitations to the order of interrupt claims and completions except when the preemptive priority mode is enabled. When PLIC is in the preemptive priority mode, the latest claimed interrupt should be completed first.

18.4.11  Preempted Priority Stack Registers for Target $m$

**Offset:** (0x200400 + 4096 * $m$) to (0x20041F + 4096 * $m$)
These registers are read/writable registers for accessing the preempted priority stack for target \( m (m \geq 0) \). These registers are used for saving and restoring priorities of the nested/preempted interrupts for a particular target by hardware. They are made available to software primarily for diagnostic purposes.

There are a total of 8 registers, each 32-bit wide, for 255 priority levels. Each bit in these registers indicates if the corresponding priority level has been preempted by a higher-priority interrupt. The location of the priority level bit for priority \( p \) of target \( m \) (Word offset Address, Bit Position) can be determined by the following equations:

- **Word offset Address**: \( 0x20_0400 + 4096 \times m + 4 \times \text{floor}(p/32) \)
- **Bit Position**: \( p \text{ modulo } 32 \)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{PL}_p )</td>
<td>([n])</td>
<td>This bit indicates that an interrupt at priority level ( p ) has been preempted by a higher-priority interrupt. The bit position ( n = p \text{ modulo } 32 ).</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupts of priority level ( p ) are preempted.</td>
</tr>
<tr>
<td>1</td>
<td>An interrupt of priority level ( p ) has been preempted.</td>
</tr>
</tbody>
</table>
19 Machine Timer

19.1 Introduction

The RISC-V architecture defines a machine timer that provides a real-time counter and generates timer interrupts. The Linux kernel expects microsecond resolutions for \texttt{mtime}, so it imposes an additional requirement that the frequency of the timer clock has to be greater than 1MHz. For non-Linux applications, the timer clock could share the same clock source as the real-time clock timer.

The RISC-V privileged specification expects that software discovers the frequency of the timer clock through a platform specific mechanism. For Linux kernels, this is achieved through the Device Tree specification.

The machine timer primarily consists of these memory-mapped registers: \texttt{mtime} and \texttt{mtimecmpn} \((n: 0 \rightarrow \text{NHART}-1)\). The \texttt{mtime} register is a 64-bit real-time counter clocked by the machine timer clock.

The \texttt{mtimecmpn} register stores a 64-bit value for comparing with \texttt{mtime}. When the value in \texttt{mtime} is greater than or equal to the value in \texttt{mtimecmpn}, the \texttt{mstatus.MTIP} timer interrupt pending bit will be set. Once set, the MTIP bit is not cleared until \texttt{mtimecmp} is written.

19.2 Machine Timer Registers

The machine timer registers are accessed through the bus interface, and their memory map is shown in Table 73.

Please note that the machine timer supports only 32-bit or 64-bit transfers. Behaviors of 8-bit and 16-bit transfers are UNDEFINED, and these transfers might be ignored as well as result in error responses or unexpected register updates.

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0 – 0x3</td>
<td>\texttt{mtime[31:0]}</td>
</tr>
<tr>
<td>0x4 – 0x7</td>
<td>\texttt{mtime[63:32]}</td>
</tr>
<tr>
<td>0x8 – 0xB</td>
<td>\texttt{mtimecmp0[31:0]}</td>
</tr>
<tr>
<td>0xC – 0xF</td>
<td>\texttt{mtimecmp0[63:32]}</td>
</tr>
<tr>
<td>0x10 – 0x13</td>
<td>\texttt{mtimecmp1[31:0]}</td>
</tr>
<tr>
<td>0x14 – 0x17</td>
<td>\texttt{mtimecmp1[63:32]}</td>
</tr>
<tr>
<td>0x18 – 0x1B</td>
<td>\texttt{mtimecmp2[31:0]}</td>
</tr>
<tr>
<td>0x1C – 0x1F</td>
<td>\texttt{mtimecmp2[63:32]}</td>
</tr>
</tbody>
</table>

Table 73: Memory Mapped Machine Timer Registers

Continued on next page…
Table 73: (continued)

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8+n<em>8 – 0xB+n</em>8</td>
<td>mtimecmp[n][31:0]</td>
</tr>
<tr>
<td>0xC+n<em>8 – 0xF+n</em>8</td>
<td>mtimecmp[n][63:32]</td>
</tr>
</tbody>
</table>

19.2.1 Machine Timer Initialization

The `mtime` counter is a 64-bit value and it increments non-stop on every machine timer clock except the first few cycles after its control register updates. When the data bus of machine timer is 64-bit, it is recommended to program the `mtime` counter through a single double-word store to guarantee the value is updated atomically. Otherwise, the following programming sequence should be followed to initialize the counter through two separate 32-bit updates to `mtime[31:0]` and `mtime[63:32]`.

- If bit[31:29] of the intended value is 7:
  1. Write zero to `mtime[31:0]`.
  2. Write high part of the intended value to `mtime[63:32]`.
  3. Write low part of the intended value to `mtime[31:0]`.

- Otherwise:
  1. Write low part of the intended value to `mtime[31:0]`.
  2. Write high part of the intended value to `mtime[63:32]`. 
20 Debug Subsystem

20.1 Overview

The AX45MP-1C debug subsystem implements RISC-V External Debug Support (TD003) V0.13. Figure 4 shows the block diagram of the debug subsystem, which contains two components: Debug Module (DM) and JTAG Debug Transport Module (JDTM). DM can be accessed through its two AHB slave ports. One is the system interface port, which is for an AndesCore processor to access the debug module through the system bus. The other one is the Debug Memory Interface (DMI) port, which is accessed by JDTM. JDTM converts debug commands in JTAG interfaces of external debuggers to bus read/write requests to the DMI port.

Debug interrupts cause AX45MP-1C to enter the debug mode and redirect the instruction fetch to the debug exception handler, whose entry point should be the base address of DebugModule.

DebugModule includes a Debug ROM at its base address that defines the debug exception handler. When invoked, the debug handler polls DebugModule internal registers to process commands issued by the external debugger through the JTAG Debug Transport module. Typical debug commands include accessing processor registers, accessing memories, and executing programs written in the Program Buffer, which is a memory region writable by the external debugger.
20.2 Optional Debug Subsystem

The debug subsystem can be removed by simply disabling the core debug feature in the configuration tool.
20.3 DebugModule

Table 74 summarizes the memory map within the DebugModule address space as viewed from the system bus interface. Please note that the offset for the program buffer could be discovered by the external debugger through execution of the AUIPC instruction as the first instruction in the program buffer, and the starting offset of Abstract Data is defined as hartinfo.DATAADDR. The offsets could be used as offsets of load/store instructions with the zero register as the base register to access this memory space. The zero register is automatically mapped to the base of DebugModule for load/store instructions in Debug Mode.

This system bus address space of DebugModule should be within a device region for the proper operation of the Debug ROM and the external debugger support.

Table 75 summarizes the memory map as viewed from the DMI interface. The address[8:2] value in the table follows the address value assignment of the debug module debug bus registers as described in RISC-V External Debug Support (TD003) V0.13.

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 – 0x007F</td>
<td>Debug ROM Internal Use Only</td>
<td></td>
</tr>
<tr>
<td>0x0080 – 0x00BF</td>
<td>Program Buffer</td>
<td>Section 20.3.12</td>
</tr>
<tr>
<td>0x00C0 – 0x00CF</td>
<td>Abstract Data 0–3</td>
<td>Section 20.3.1</td>
</tr>
<tr>
<td>0x00D0 – 0x01FF</td>
<td>Reserved for internal use N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address[8:2]</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04 – 0x07</td>
<td>Abstract Data 0–3</td>
<td>Section 20.3.1</td>
</tr>
<tr>
<td>0x10</td>
<td>Debug Module Control</td>
<td>Section 20.3.2</td>
</tr>
<tr>
<td>0x11</td>
<td>Debug Module Status</td>
<td>Section 20.3.3</td>
</tr>
<tr>
<td>0x12</td>
<td>Hart Info</td>
<td>Section 20.3.4</td>
</tr>
<tr>
<td>0x13</td>
<td>Halt Summary</td>
<td>Section 20.3.5</td>
</tr>
<tr>
<td>0x14</td>
<td>Hart Array Window Select</td>
<td>Section 20.3.6</td>
</tr>
<tr>
<td>0x15</td>
<td>Hart Array Window</td>
<td>Section 20.3.7</td>
</tr>
<tr>
<td>0x16</td>
<td>Abstract Control and Status</td>
<td>Section 20.3.8</td>
</tr>
<tr>
<td>0x17</td>
<td>Abstract Command</td>
<td>Section 20.3.9</td>
</tr>
<tr>
<td>0x18</td>
<td>Abstract Command Autoexec</td>
<td>Section 20.3.10</td>
</tr>
<tr>
<td>0x19 – 0x1C</td>
<td>Device Tree Addr 0–3</td>
<td>Section 20.3.11</td>
</tr>
</tbody>
</table>

Continued on next page...
### Table 75: (continued)

<table>
<thead>
<tr>
<th>Address[8:2]</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20 – 0x2F</td>
<td>Program Buffer 0–15</td>
<td>Section 20.3.12</td>
</tr>
<tr>
<td>0x30</td>
<td>Authentication Data</td>
<td>Section 20.3.13</td>
</tr>
<tr>
<td>0x38</td>
<td>System Bus Access Control and Status</td>
<td>Section 20.3.14</td>
</tr>
<tr>
<td>0x39 – 0x3B</td>
<td>System Bus Address</td>
<td>Section 20.3.15</td>
</tr>
<tr>
<td>0x3C – 0x3F</td>
<td>System Bus Data</td>
<td>Section 20.3.16</td>
</tr>
</tbody>
</table>
20.3.1 Abstract Data 0–3 (data0 – data3)

Basic read/write registers that may be read or changed by abstract commands.

The registers are accessible from both the DMI interface and the system bus interface to support data exchanges between the external debugger and the processor (i.e., instructions in the program buffer).

20.3.2 Debug Module Control (dmcontrol)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmactive</td>
<td>[0]</td>
<td>Controlling reset signal for Debug Module itself.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>The Debug Module’s state takes its reset values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>The Debug Module functions normally.</td>
<td></td>
</tr>
<tr>
<td>ndmreset</td>
<td>[1]</td>
<td>Controlling reset signal from the Debug Module to the rest of the system.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Deassert system reset signal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Assert system reset signal.</td>
<td></td>
</tr>
<tr>
<td>clrresethaltreq</td>
<td>[2]</td>
<td>This optional field clears the halt-on-reset request bit for all currently selected harts. Writes apply to the new value of hartsel and hasel.</td>
<td>W1</td>
<td>-</td>
</tr>
</tbody>
</table>

Continued on next page...
Field Name | Bits | Description | Type | Reset
--- | --- | --- | --- | ---
setresethaltreq | [3] | This optional field writes the halt-on-reset request bit for all currently selected harts, unless clrresethaltreq is simultaneously set to 1. When set to 1, each selected hart will halt upon the next deassertion of its reset. The halt-on-reset request bit is not automatically cleared. The debugger must write to clrresethaltreq to clear it. Writes apply to the new value of hartsel and hasel. If hasresethaltreq is 0, this field is not implemented. | W1 | -

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There is a single currently selected hart, that is selected by hartsel.</td>
</tr>
<tr>
<td>1</td>
<td>There may be multiple currently selected harts selected by hartsel, plus those selected by the hart array mask register.</td>
</tr>
</tbody>
</table>

hartsel | [25:16] | Selecting the target hart to be debugged. | RW | 0x0

hasel | [26] | Selects the definition of currently selected harts. | RW | 0x0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There is a single currently selected hart, that is selected by hartsel.</td>
</tr>
<tr>
<td>1</td>
<td>There may be multiple currently selected harts selected by hartsel, plus those selected by the hart array mask register.</td>
</tr>
</tbody>
</table>

ackhavereset | [28] | Writing 1 to this bit clears the havereset bits for any selected harts. Harts are selected based on the new value of hartsel and hasel being written. | W1 | 0x0

resumereq | [30] | Writes the resume request bit for all currently selected harts. When set to 1, each selected hart will resume if it is currently halted. The resume request bit is ignored while the halt request bit is set. Harts are selected based on the new value of hartsel and hasel being written. | WO | 0x0

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>haltreq</td>
<td>[31]</td>
<td>Writes the halt request bit for all currently selected harts. When set to 1, each selected hart will halt if it is not currently halted. Writing 1 or 0 has no effect on a hart which is already halted, but the bit must be cleared to 0 before the hart is resumed. Harts are selected based on the new value of hartsel and hasel being written.</td>
<td>WO</td>
<td>0x0</td>
</tr>
</tbody>
</table>

**Note**

- DebugModule uses wraparound hart ID to select harts, which can reduce the integration effort on a multi-hart system with multiple debug modules. When a hart ID is larger than the effective bit value, it will be mapped to a smaller value. For example, hart 10 will be remapped to hart 2 on a debug module instance supporting 8 harts (NHART=8). Essentially, the effective bits of hart ID used in DebugModule is the ceiling of log2 value of NHART.
20.3.3 Debug Module Status (dmstatus)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>[3:0]</td>
<td>Version of the implemented RISC-V External Debug Support. 0x2 indicates that the current implemented version is 0.13.</td>
<td>RO</td>
<td>0x2</td>
</tr>
<tr>
<td>devtreevalid</td>
<td>[4]</td>
<td>Whether the information in devtreeaddr0 – devtreeaddr3 registers holds the address of the Device Tree.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>hasresethaltreq</td>
<td>[5]</td>
<td>This bit indicates the supported status of resethaltreq.</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>authbusy</td>
<td>[6]</td>
<td></td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>The authentication module is ready to process the next read/write to authdata.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The authentication module is busy.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>authenticated</td>
<td>[7]</td>
<td></td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Authentication is required before using the Debug Module.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Authentication check has passed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anyhalted</td>
<td>[8]</td>
<td>Indicates whether any currently selected hart is halted.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allhalted</td>
<td>[9]</td>
<td>Indicates whether all currently selected harts are halted.</td>
<td>RO</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyrunning</td>
<td>[10]</td>
<td>Indicates whether any currently selected hart is running.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allrunning</td>
<td>[11]</td>
<td>Indicates whether all currently selected harts are running.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>anyunavail</td>
<td>[12]</td>
<td>Indicates whether any currently selected hart is unavailable.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allunavail</td>
<td>[13]</td>
<td>Indicates whether all currently selected harts are unavailable.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>anynonexistent</td>
<td>[14]</td>
<td>Indicates whether any currently selected hart does not exist in this system.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allnonexistent</td>
<td>[15]</td>
<td>Indicates whether all currently selected harts do not exist in this system.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>anyresumeack</td>
<td>[16]</td>
<td>Indicates whether any currently selected hart has acknowledged the previous resume request.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allresumeack</td>
<td>[17]</td>
<td>Indicates whether all currently selected harts have acknowledged the previous resume request.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>anyhavereset</td>
<td>[18]</td>
<td>Indicates whether any currently selected hart has been reset but the reset has not been acknowledged</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>allhavereset</td>
<td>[19]</td>
<td>Indicates whether all currently selected harts have been reset but the reset has not been acknowledged</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>impebreak</td>
<td>[22]</td>
<td>Indicates whether there is an implicit EBREAK instruction at the nonexistent word immediately after the program buffer.</td>
<td>RO</td>
<td>0x1</td>
</tr>
</tbody>
</table>

**Note**

- The reset values with * mark may not reflect the transient hart states when DebugModule is under reset or dmcontrol.DMACTIVE is not set.
20.3.4 Hart Info (hartinfo)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataaddr</td>
<td>[11:0]</td>
<td>Signed offset for accessing the shadowed data registers by the processor, to be used as offsets</td>
<td>RO</td>
<td>0xC0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for load/store instructions with the zero register as the base register in Debug Mode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datasize</td>
<td>[15:12]</td>
<td>Number of 32-bit words in the memory map dedicated to shadowing the data registers.</td>
<td>RO</td>
<td>0x4</td>
</tr>
<tr>
<td>dataaccess</td>
<td>[16]</td>
<td>The method for accessing the shadowed data registers. The value of this field is 0x1 for</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AX45MP-1C, indicating that the data registers are shadowed in the memory map of the selected hart under Debug Mode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nscratch</td>
<td>[23:20]</td>
<td>Number of dscratch registers available for the debugger to use during program buffer execution, starting from dscratch0.</td>
<td>RO</td>
<td>0x2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The data registers are shadowed in the hart by CSR registers.</td>
</tr>
<tr>
<td>1</td>
<td>The data registers are shadowed in the hart's memory map. Each register takes up 4 bytes in the memory map.</td>
</tr>
</tbody>
</table>
20.3.5 Halt Summary (haltsum)

Bit 0 contains the logical OR of 32 halt bits for hart 0 – hart 31.

20.3.6 Hart Array Window Select (hawindowsel)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>hawindowsel</td>
<td>[14:0]</td>
<td>This register selects which of the 32-bit portion of the hart array mask register is accessible in hawindow.</td>
<td>RW</td>
<td>0x0</td>
</tr>
</tbody>
</table>

20.3.7 Hart Array Window (hawindow)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>hawindow</td>
<td>[31:0]</td>
<td>This register provides R/W accesses to a 32-bit portion of the hart array mask register. The position of the window is determined by hawindowsel. That is, bit 0 refers to hart (hawindowsel * 32), while bit 31 refers to hart (hawindowsel * 32 + 31).</td>
<td>RW</td>
<td>0x0</td>
</tr>
</tbody>
</table>
20.3.8 Abstract Control and Status (abstractcs)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>datacount</td>
<td>[4:0]</td>
<td>Number of data registers that are implemented.</td>
<td>RO</td>
<td>0x4</td>
</tr>
<tr>
<td>cmderr</td>
<td>[10:8]</td>
<td>Error code indicating that an abstract command fails. The bits in this field remain set until they are cleared by writing 1 to them. No abstract command is started until the value is reset to 0.</td>
<td>R/W1C</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None: no error</td>
</tr>
<tr>
<td>1</td>
<td>Busy: an abstract command was executing while command, abstractcs, or abstractauto was written, or when one of the data or progbuf registers was read or written.</td>
</tr>
<tr>
<td>2</td>
<td>Not supported: the requested command is not supported.</td>
</tr>
<tr>
<td>3</td>
<td>Exception: an exception occurred while executing the command.</td>
</tr>
<tr>
<td>4</td>
<td>Halt/resume: an abstract command cannot execute because the hart wasn’t in the expected state (running/halted).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>[12]</td>
<td>Flag indicating an abstract command is currently being executed.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>progbufsize</td>
<td>[28:24]</td>
<td>Size of the program buffer, in 32-bit words.</td>
<td>RO</td>
<td>0x8</td>
</tr>
</tbody>
</table>

20.3.9 Abstract Command (command)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdtype</td>
<td>[31:24]</td>
<td></td>
</tr>
<tr>
<td>control</td>
<td>[23:20]</td>
<td></td>
</tr>
</tbody>
</table>
Field Name | Bits    | Description                                                                 | Type | Reset |
-----------|---------|-----------------------------------------------------------------------------|------|-------|
control    | [23:0]  | The field is interpreted in a command-specific manner.                     | WO   | 0x0   |
cmdtype    | [31:24] | Controlling the overall functionality of this abstract command.            | WO   | 0x0   |

If a command takes arguments or returns values, they must be written to or placed in the data registers. Which data registers are used for the arguments is described in Table 76.

Table 76: Use of Data Registers in PLDM

<table>
<thead>
<tr>
<th>Argument Width</th>
<th>arg0/Return Value</th>
<th>arg1</th>
<th>arg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>data0</td>
<td>data1</td>
<td>data2</td>
</tr>
<tr>
<td>64</td>
<td>data0, data1</td>
<td>data2, data3</td>
<td>data4, data5</td>
</tr>
</tbody>
</table>

20.3.9.1 Access Register

This command gives the debugger access to CPU registers and allows CPU to execute the program buffer.

Field Name | Bits    | Description                                                                 | Type | Reset |
-----------|---------|-----------------------------------------------------------------------------|------|-------|
regno      | [15:0]  | Number of the specified register to be accessed.                           | WO   | 0x0   |
write      | [16]    | The direction of data transfer.                                             | WO   | 0x0   |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Copy data from the specified register into arg0 portion of the Abstract Data registers.</td>
</tr>
<tr>
<td>1</td>
<td>Copy data from arg0 portion of Abstract Data registers into the specified register.</td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>transfer</td>
<td>[17]</td>
<td>Indicates whether to perform data transfer.</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Don’t do the operation specified by write.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do the operation specified by write.</td>
<td></td>
</tr>
<tr>
<td>postexec</td>
<td>[18]</td>
<td>Indicates whether to execute the program in the program buffer.</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this field is set, execute the program in the Program Buffer exactly once after performing the transfer, if any.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>[22:20]</td>
<td>Value</td>
<td>Meaning</td>
<td>WO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Access the lowest 32 bits of the register</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Access the lowest 64 bits of the register</td>
<td></td>
</tr>
</tbody>
</table>

### 20.3.9.2 Quick Access

Perform the following sequence of operations:

- If the hart is halted already, the command sets `cmderr` to `halt/resume` and does not continue.
- Halt the hart. If the hart halts for some other reason (e.g., breakpoint), the command sets `cmderr` to `halt/resume` and does not continue.
- Execute the program buffer. If an exception occurs, `cmderr` is set to exception and the program buffer execution ends, but the quick access command continues.
- Resume the hart.
20.3.9.3 Access Memory

This command lets the debugger perform memory accesses with the memory view of the selected hart.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>[16]</td>
<td>The direction of data transfer.</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Copy data from the memory location into arg0 portion of the Abstract Data registers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Copy data from arg0 portion of Abstract Data registers into the memory location.</td>
<td></td>
</tr>
<tr>
<td>aampostincrement</td>
<td>[19]</td>
<td>Increment arg1 by the number of bytes encoded in aamsize after a memory access completes.</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td>aamsize</td>
<td>[22:20]</td>
<td>Size of memory accesses.</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Access the lowest 8 bits of the memory location.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Access the lowest 16 bits of the memory location.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Access the lowest 32 bits of the memory location.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Access the lowest 64 bits of the memory location.</td>
<td></td>
</tr>
<tr>
<td>aamvirtual</td>
<td>[23]</td>
<td>Virtual or physical address accesses</td>
<td>WO</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Addresses are physical</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>No action</td>
<td></td>
</tr>
</tbody>
</table>

The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.
20.3.10  Abstract Command Autoexec (abstractauto)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoexecdata</td>
<td>[3:0]</td>
<td>When a bit in this field is 1, read or write accesses to the corresponding data word cause the command in command to be executed again.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td>autoexecprogbuf</td>
<td>[23:16]</td>
<td>When a bit in this field is 1, read or write accesses to the corresponding progbuf word cause the command in command to be executed again.</td>
<td>RW</td>
<td>0x0</td>
</tr>
</tbody>
</table>

20.3.11  Device Tree Addr 0–3 (devtreeaddr0 – devtreeaddr3)

The devicetreeaddr registers are hardwired to zeros in DebugModule.

20.3.12  Program Buffer 0–15 (progbuf0 – progbuf15)

The progbuf registers provide read/write accesses to the program buffer. DebugModule supports program buffer 0–7 only. Program buffer 8–15 are hardwired to 0x00100073 (the EBREAK instruction).

These registers are read/write accessible from both the DMI interface and the system bus, in addition to being a valid region for instruction fetches. They hold small programs written by the external debugger and these small programs will be fetched and executed by the processor upon execution of the abstract commands which require execution of the program buffer.

Programs in progbuf must end with EBREAK or C.EBREAK instructions if the program size is less than 8 words (32 bytes).

20.3.13  Authentication Data (authdata)

The authdata register is hardwired to zeros in DebugModule.
20.3.14 System Bus Access Control and Status (sbcs)

The `sbcs` register holds the control bits and status flags of System Bus Accesses. It is only valid when the `SYSTEM_BUS_ACCESS_SUPPORT` parameter of DebugModule is set. It is otherwise hardwired to zero.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbaccess8</td>
<td>[0]</td>
<td>This bit indicates the supported status of 8-bit system bus data accesses.</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>sbaccess16</td>
<td>[1]</td>
<td>This bit indicates the supported status of 16-bit system bus data accesses.</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>sbaccess32</td>
<td>[2]</td>
<td>This bit indicates the supported status of 32-bit system bus data accesses.</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>sbaccess64</td>
<td>[3]</td>
<td>This bit indicates the supported status of 64-bit system bus data accesses.</td>
<td>RO Dependent</td>
<td>Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This bit is 1 if <code>SYS_DATA_WIDTH == 64</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sbaccess128</td>
<td>[4]</td>
<td>This bit indicates the supported status of 128-bit system bus data accesses.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>sbasize</td>
<td>[11:5]</td>
<td>Address width of System bus addresses in bits. This field is 0 if there is no bus access support. Otherwise, it is the value of the <code>SYS_ADDR_WIDTH</code> parameter of DebugModule.</td>
<td>RO Dependent</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>sberror</td>
<td>[14:12]</td>
<td>Error code indicating the failure type of the system bus accesses. Write 1 to clear the status.</td>
<td>R/W1C</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>No bus error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Timeout error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Bad address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Alignment error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Unsupported size was requested</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>sbreadondata</td>
<td>[15]</td>
<td>Every read from sbdata0 automatically triggers a system bus read at the (possibly auto-incremented) address.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td>sbautoincrement</td>
<td>[16]</td>
<td>sbaddress is incremented by the size specified in sbaccess after every system bus access.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td>sbaccess</td>
<td>[19:17]</td>
<td>Access size.</td>
<td>RW</td>
<td>0x2</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
<td><strong>Meaning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>16-bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32-bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>64-bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>128-bit</td>
<td></td>
</tr>
<tr>
<td>sbreadonaddr</td>
<td>[20]</td>
<td>Every write to sbaddress0 automatically triggers a system bus read at the new address.</td>
<td>RW</td>
<td>0x0</td>
</tr>
<tr>
<td>sbbusy</td>
<td>[21]</td>
<td>Indicate the system bus master is busy.</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>sbbusyerror</td>
<td>[22]</td>
<td>Indicate the debugger attempts to execute system bus access before sbbusy is cleared.</td>
<td>R/W1C</td>
<td>0x0</td>
</tr>
<tr>
<td>sbversion</td>
<td>[31:29]</td>
<td>The version of the supported System Bus Access version. It is currently 1 for v0.13 of the RISC-V External Debug Support specification.</td>
<td>RO</td>
<td>0x1</td>
</tr>
</tbody>
</table>
Note
The states of \textit{sbaccess8} – \textit{sbaccess128} are set based on the \texttt{SYS_ADDR_WIDTH} parameter in DebugModule.

20.3.15 System Bus Address (\textit{sbaddress0} – \textit{sbaddress2})

The \textit{sbaddress} registers are only valid when the \texttt{SYSTEM_BUS_ACCESS_SUPPORT} parameter of DebugModule is set. They are otherwise hardwired to zeros.

<table>
<thead>
<tr>
<th>Address Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{sbaddress0}</td>
<td>\texttt{bit[31:0]} of the address in \textit{sbaddress}</td>
</tr>
<tr>
<td>\textit{sbaddress1}</td>
<td>\texttt{bit[63:32]} of the address in \textit{sbaddress}</td>
</tr>
<tr>
<td>\textit{sbaddress2}</td>
<td>\texttt{bit[95:64]} of the address in \textit{sbaddress}</td>
</tr>
<tr>
<td>\textit{sbaddress3}</td>
<td>\texttt{bit[127:96]} of the address in \textit{sbaddress}</td>
</tr>
</tbody>
</table>

20.3.16 System Bus Data (\textit{sbdata0} – \textit{sbdata3})

The \textit{sbdata} register is supported as below when \texttt{SYSTEM_BUS_ACCESS_SUPPORT} parameter of DebugModule is set. Otherwise it is hardwired to zeros.

<table>
<thead>
<tr>
<th>Address Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{sbdata0}</td>
<td>\texttt{bit [31:0]} of the address in \textit{sbdata}</td>
</tr>
<tr>
<td>\textit{sbdata1}</td>
<td>\texttt{bit [63:32]} of the address in \textit{sbdata}</td>
</tr>
<tr>
<td>\textit{sbdata2}</td>
<td>\texttt{bit [95:64]} of the address in \textit{sbdata}</td>
</tr>
<tr>
<td>\textit{sbdata3}</td>
<td>\texttt{bit [127:96]} of the address in \textit{sbdata}</td>
</tr>
</tbody>
</table>

20.4 JTAG Debug Transport Module

JDTM is an implementation of the JTAG debug transport module (DTM), as defined by the spec: \textit{RISC-V External Debug Support (TD003) V0.13}. It implements the IEEE 1149.1 style test access port controller (TAP). The supported instructions are summarized in Table 79.
Table 79: Supported TAP Instructions of JDTM

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Instruction</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11111</td>
<td>BYPASS</td>
<td>Section 20.4.1</td>
</tr>
<tr>
<td>b00001</td>
<td>IDCODE</td>
<td>Section 20.4.2</td>
</tr>
<tr>
<td>b10000</td>
<td>dtmcs</td>
<td>Section 20.4.3</td>
</tr>
<tr>
<td>b10001</td>
<td>dmi</td>
<td>Section 20.4.4</td>
</tr>
</tbody>
</table>

20.4.1 BYPASS

When the TAP instruction is BYPASS, a single-bit register is connected to tdi and tdo. In Capture-DR state, the register is loaded by 0. In Shift-DR state, data is transferred from tdi to tdo through the single-bit register.

20.4.2 IDCODE

This register contains device identification code: 0x1000563D.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManufId</td>
<td>[11:1]</td>
<td>Identifies the designer/manufacturer of this part.</td>
<td>RO</td>
<td>0x31E</td>
</tr>
<tr>
<td>PartNumber</td>
<td>[27:12]</td>
<td>Identifies the designer’s part number of this part.</td>
<td>RO</td>
<td>0x0005</td>
</tr>
<tr>
<td>Version</td>
<td>[31:28]</td>
<td>Identifies the release version of this part.</td>
<td>RO</td>
<td>0x1</td>
</tr>
</tbody>
</table>
### 20.4.3 DTM Control and Status (dtmcs)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>[3:0]</td>
<td>Version of the implemented DTM. 0x1 indicates that the current implementation conforms to RISC-V External Debug Support (TD003) V0.13.</td>
<td>RO</td>
<td>0x1</td>
</tr>
<tr>
<td>abits</td>
<td>[9:4]</td>
<td>Bit width of DMI address is 7</td>
<td>RO</td>
<td>0x7</td>
</tr>
<tr>
<td>dmistat</td>
<td>[11:10]</td>
<td>State of DMI</td>
<td>RO</td>
<td>0x0</td>
</tr>
<tr>
<td>idle</td>
<td>[14:12]</td>
<td>This is a hint to the debugger of the minimum number of cycles a debugger should spend in RunTest/Idle after every DMI scan to avoid a busy return code (dmistat of 3).</td>
<td>RO</td>
<td>0x7</td>
</tr>
<tr>
<td>dmireset</td>
<td>[16]</td>
<td>Writing 1 to this bit clears the sticky error state and allows the DTM to retry or complete the previous transaction.</td>
<td>W1</td>
<td>0</td>
</tr>
<tr>
<td>dmihardreset</td>
<td>[17]</td>
<td>Writing 1 to this bit does a hard reset of the DTM, causing the DTM to forget about any outstanding DMI transactions. In general, this should only be used when the debugger has reasons to expect that the outstanding DMI transaction will never complete (e.g., a reset condition causes an inflight DMI transaction to be canceled).</td>
<td>W1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>An operation failed (resulted in op of 2)</td>
</tr>
<tr>
<td>3</td>
<td>An operation was attempted while a DMI access was still in progress (resulted in op of 3)</td>
</tr>
</tbody>
</table>
20.4.4 Debug Module Interface Access (dmi)

The Debug Module Interface (DMI) is accessed through this register.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
<th>Type</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>[1:0]</td>
<td>Write operation:</td>
<td>RW</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Ignore data and address. (nop)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Read from address. (read)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Write data to address. (write)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read operation:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>The previous operation completed successfully.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A previous operation failed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>An operation was attempted while a DMI request is still in progress. The data scanned into dmi in this access will be ignored.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>[33:2]</td>
<td>The data to send to the DM over the DMI during Update-DR, and the data returned from the DM as a result of the previous operation.</td>
<td>RW</td>
<td>0</td>
</tr>
<tr>
<td>address</td>
<td>[40:34]</td>
<td>Address used for DMI access. In Update-DR this value is used to access the DM over the DMI.</td>
<td>RW</td>
<td>0</td>
</tr>
</tbody>
</table>