# Arm® Generic Interrupt Controller Architecture Specification

GIC architecture version 5

| | |
|---|---|
| Document number | ARM-AES-0070 |
| Document quality | EAC |
| Document version | 00eac0 |
| Document confidentiality | Non-confidential |

*opyright 2022-202 m Limited or its affiliates. All rights reserved.*

RETIRED

**This document is now RETIRED.**
**The Arm Generic Interrupt Controller Architecture Specification (IHI111701) is the definitive reference for this architecture specification.**

# Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5

## Release information

| Date | Version | Changes |
|---|---|---|
| 2025/Sep/10 | 00eac0 | • [GICSPEC-680] Architected mechanism to mange PE to IRS connection<br>• [AARCH-24093] Remove section describing interaction with FEAT_TME.<br>• [GICSPEC-746] Clarify how GIC Stream rules might appear to a protocol checker.<br>• [GICSPEC-700] Make CoreSight registers for different frames more easily searchable.<br>• [GICSPEC-754] Software is required to supp[illegible]1111 for GSB.<br>• [GICSPEC-753] Correct condition for IR[illegible]PE_CR0.D[illegible]<br>• [GICSPEC-755] Correct definition of N[illegible]<br>• [GICSPEC-756] Add definition of W1C to [illegible]mory-map[illegible]d register access permissions.<br>• [GICSPEC-758] Clarify requi[illegible]ments w[illegible]n SetR[illegible]nt Ack is sent early.<br>• [GICSPEC-746] Clarify beha[illegible]r wh[illegible] resetting an online Interrupt Signaling Channel.<br>• [GICSPEC-759] Cla[illegible] I[illegible]rupt S[illegible]naling [illegible]hannel rules.<br>• [GICSPEC-757] C[illegible]rify wh[illegible]h domai[illegible]s are assigned to at reset. |
| 2025/Aug/15 | 00bet2 | • [GICSPEC-737] The IWB[illegible]WDOMAINR register array base address is updated to address 0x8000.<br>• [GICSPEC-[illegible]1] Co[illegible]t length of VPE field in SetResident to be 16 bits wide.<br>• [GICS[illegible]EC-72[illegible] Restric[illegible]hen the IRS can send Forward commands while making a VP[illegible]esident.<br>• [GIC[illegible]PEC-735] The[illegible]ffective value of ICC_SRE_EL3.SRE should be 1 when FEAT_G[illegible]IE is implemented.<br>• [GICSPEC[illegible]33] Clarify that access to GICv3 ICC_ registers is UNDEFINED.<br>• [GICSPEC-74[illegible]] Check for IRS_SYNCR_STATUSR.IDLE has incorrect polarity in [illegible]RS synchronization request ordering rule.<br>[GICSPEC-744] Ordering and synchronization requirements of GIC PMU events.<br>• [GIC[illegible]PEC-738] Accesses to GICv3 SGI registers trap when ICH_HCR_EL2.TC is se[illegible]o 1 in Legacy operation.<br>• [GICSPEC-722] Allow Release commands to be sent while RequestConfig is outstanding.<br>• [GICSPEC-688] Clarify handling of TKEEP and TLAST.<br>• [GICSPEC-705] Clarify that replaced interrupts are returned in finite time.<br>• [GICSPEC-728] Make IRS_VPE_CR0.DPS RES0 when VIRT_ONE_N is not implemented.<br>• [GICSPEC-734] Other IST-related fields in IRS_IDR2 should not be RES0 when IRS_IDR2.LPI is 0.<br>• [GICSPEC-736] Clarify which GIC instructions fail silently for unreachable INTIDs.<br>• [GICSPEC-747] Clarify that IRS SWERR codes apply to writes as well as reads.<br>• [GICSPEC-741] Simplify and clarify Superpriority rules. |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Date | Version | Changes |
|---|---|---|
| 2025/Jul/14 | 00bet1 | • Second BET release.<br>• Support for 4 bits of priority in the CPU interface has been removed.<br>• The GIC Stream reset sequences are modified to include a sequence number. The behavior relying on the sequence number is described in sections A4.3 and A5.4.<br>• In section B1.42, new litmus tests are added to illustrate effects on the HPPI when writing to ICx_PCR_ELx.<br>• In section 10.6, a DEVARCH value has been defined for the CoreSight identification registers.<br>• In section 9.6.2, the description for the behavior of the ICH_CONTEXTR_EL2 the V and F fields is clarified.<br>• In section 10.2.1.21, the IRS Domain used when accessing IRS_PE_CR0 is updated to be specific to the frame being accessed.<br>• In section 2.10.1.2, incorrect register references to ICC and ICV _EL2 registers are corrected to reference the ICC and ICV _EL1 registers.<br>• In section 2.6, the effects on asynchronous exceptions when executing a GSB are clarified.<br>• In section 9.7.5, all fields in ICH_HCR_EL2 other than ICH_HCR_EL2.EN do not need an Effective value when legacy operation is disabled.<br>• In section 2.10.6, the behavior of writes to ICH_CONTEXTR_EL2 to request a VPE doorbells is clarified.<br>• In section 2.7 describing Interrupt Prioritization, a rule is added to specify the required effects of PPI updates on the asynchronous exceptions following a context synchronization event.<br>• In section 9.1 describing the behavior of the Legacy virtual CPU interface registers, the access behaviors for ICC_SRE_ELx are corrected to support nested virtualization.<br>• In section 4.10.7 describing the IRS VPE doorbell handling, it is clarified that IRS_VPE_DBR.RESDB is cleared when generating a doorbell event.<br>• In sections 10.2.1.12, 10.3.1.15, and 10.4.1.3, the descriptions of the Implementor field in the IRS_IIDR, ITS_IIDR, and IWB_IIDR registers are clarified.<br>• In section 9.1 the use of self-synchronizing to describe the behavior of GICv5 system registers is replaced with a description of when the effects from accesses to the registers are visible to other instructions.<br>• In section 2.7, the rules for completion of updates to PPI state and configuration are clarified.<br>In section 10.2.1.25 the polarity of IRS_SAVE_VM_STATUSR.Q is corrected.<br>• In section 2.9.1, a rule is added describing the results of not implementing a PPI. Additionally, in section 2.10.7, a rule is added to specify that the GIC maintenance interrupt is only implemented when support for Legacy operation is implemented.<br>• In section A5.4, describing the GIC Stream protocol, an incorrect reference to the Interrupt Handling channel is corrected to reference the Interrupt Signaling channel.<br>• In section 9.7 describing the Legacy hypervisor control registers, the behavior is corrected such that the legacy hypervisor control registers are accessible when ICH_VCTLR_EL2.V3 is 1 and UNDEFINED otherwise. Additionally, in section 8.4 describing the GSB instruction, the instruction behavior is corrected such that it is UNDEFINED when ICH_VCTLR_EL2.V3 is 1.<br>• In section 10.2.1.8, the minimum and maximum values for IRS_IDR4.VPE_ID_BITS are corrected.<br>• In section 9.6.2 describing the ICH_CONTEXTR_EL2 System register, the behavior of writes to the register is clarified. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| Date | Version | Changes |
|---|---|---|
| 2025/Jul/14 | 00bet1 | • In section A4.4.2 describing the GIC Stream protocol, it is clarified to better explain the behavior when there is no resident VPE.<br>• In section 8.3.2, an error is corrected in the description of the GIC LDDI instruction.<br>• In sections 8.2.2 and 8.2.5 describing the GIC VDDI and GIC VDHM instructions, the accessor pseudocode is corrected so that the instructions are UNDEFINED when EL2 is not enabled.<br>• In section 9.4.3 describing the ICC_PPI_DOMAINR_EL3 registers, the text that says that undefined Interrupt Domain encodings are reserved is removed. In section 2.9.1 describing the overall behavior of Physical PPIs, an info statement is added to describe the behavior of PPIs when they are assigned to an Interrupt Domain encoding for an Interrupt Domain that is not implemented.<br>• In section 4.5 describing the IRS synchronizati[illegible]uests, it is clarified that IRS synchronization requests are global to all I[illegible]s for the [illegible]errupt Domain.<br>• In section 4.5 describing the IRS synchro[illegible]zation requests, SPIs are removed from the interrupt effects synchronized by an IRS s[illegible]hronization [illegible]quest.<br>• In section A2.2 describing the GI[illegible]tream pr[illegible]col, an i[illegible]correct reference to the IDDTID signal is corrected to [illegible]erence the ICC[illegible]D [illegible]gnal.<br>• In section A4.6 describing t[illegible] GIC Str[illegible]m protocol, the Activate command is added to the list of commands whose [illegible]fec[illegible] are globa[illegible]y observed when the command is acknowledged.<br>• In section A4.7 des[illegible]ibing t[illegible] GIC St[illegible]m protocol, the reference to an INTID in the Recall comman[illegible]s remove[illegible].<br>• In section A4.6 d[illegible]scribi[illegible]g the G[illegible]tream protocol, the SetHandling command is removed fr[illegible] the co[illegible]tions necessary for the IRS to send Sync Ack.<br>• In secti[illegible]n 8.2.[illegible] the VM[illegible]eld of the GIC VDPEND instruction is expanded to be 16 bits[illegible]de coverin[illegible]its [47[illegible]2] and the PENDING field is moved to bit [63].<br>• I[illegible]se[illegible]ion 4.8.4 desc[illegible]bing the IRS Physical interrupt signaling, it is clarified that changes[illegible]f priority to active interrupts do not affect Running Priority. |
| 2025/Mar/28 | 00bet0 | • First BET[illegible]lease. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Non-Confidential Proprietary Notice

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

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

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

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

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

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

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

The validity, construction and performance of this notice shall be governed by English Law.

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

Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-20349

8 March 2024

RETIRED

### *Confidentiality Status*

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

### *Product Status*

The information in this document is at EAC quality. EAC quality means that all features of the specification are described in the manual.

### *Web Address*

https://www.arm.com

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## Contents

# Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5


Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5 ii
Release information . . . ii
Non-Confidential Proprietary Notice . . . v

**Preface**

Conventions . . . xviii
Typographical conventions . . . xviii
Numbers . . . xviii
Pseudocode descriptions . . . xviii
Assembler syntax descriptions . . . xviii
Rules-based writing . . . xix
Content item identifiers . . . xix
Content item rendering . . . xix
Content item classes . . . xix
Additional reading . . . xx
Feedback . . . xxii
Feedback on this book . . . xxii
Inclusive terminology commitment . . . xxii

**Chapter 1 Introduction**

**Chapter 2 PE architecture**

2.1 Architecture features and extensions . . . 25
2.2 The GICv5 CPU interface . . . 27
2.2.1 IRI link management . . . 28
2.3 Interrupt Domains . . . 30
2.4 Interrupt types and identifiers . . . 32
2.4.1 PE-Private Peripheral Interrupts (PPIs) . . . 33
2.4.2 Logical Peripheral Interrupts (LPIs) . . . 34
2.4.3 Shared Peripheral Interrupts (SPIs) . . . 35
2.5 Inter-Processor Interrupts . . . 36
2.6 GIC System instructions . . . 38
2.6.1 GIC synchronization barriers (GSBs) . . . 40
2.6.2 LPI and SPI configuration . . . 40
2.7 Interrupt Prioritization . . . 42
2.8 Interrupt handling . . . 44
2.8.1 Interrupt life cycle . . . 45
2.9 The physical CPU interface . . . 48
2.9.1 Physical PPIs . . . 48
2.9.2 Physical priority masking . . . 51
2.9.3 Preemptive interrupts . . . 51
2.9.4 Physical interrupt signaling . . . 52
2.9.5 Physical non-maskable interrupts . . . 55
2.9.6 Doorbell PPIs . . . 56
2.10 The virtual CPU interface . . . 59
2.10.1 Virtual PPIs . . . 59
2.10.2 Virtual priority masking . . . 62


RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

2.10.3 Virtual interrupt signaling . . . 63
2.10.4 Virtual non-maskable interrupts . . . 65
2.10.5 Selecting the resident VPE . . . 65
2.10.6 Requesting VPE doorbells . . . 67
2.10.7 Legacy virtual CPU interface . . . 68
2.11 GIC synchronous exception priorities . . . 70
2.12 Interrupt ordering model and synchronization requirements . . . 71
2.12.1 GIC and GICR ordering semantics . . . 72
2.12.2 GSB instruction semantics . . . 74
2.12.3 GIC Ordering Model . . . 74

**Chapter 3 GICv5 system architecture**
3.1 Interrupt Domains . . . 82
3.2 Communication between GIC system components . . . 84
3.3 Coherency considerations for GIC data structures . . . 85

**Chapter 4 Interrupt routing service (IRS)**
4.1 Communication between the IRS and the CPU interface . . . 88
4.2 Signaling interrupts . . . 89
4.3 IRS Domains . . . 91
4.4 IRS Configuration . . . 93
4.4.1 Enabling and disabling the IRS . . . 95
4.5 IRS synchronization requests . . . 96
4.6 Interrupt configuration and state . . . 98
4.7 The interrupt state table (IST) . . . 100
4.7.1 Level 2 IST management . . . 102
4.7.2 Initialization of level 2 IST entries . . . 104
4.7.3 IAFFID state and configuration . . . 104
4.7.4 IST metadata . . . 105
4.7.5 Example IST structures . . . 106
4.8 Physical interrupts . . . 108
4.8.1 Physical LPIs . . . 108
4.8.2 Physical SPIs . . . 109
4.8.3 Physical interrupt routing . . . 114
4.8.4 Physical interrupt signaling . . . 115
4.9 Virtualization data structures . . . 118
4.9.1 The VM table . . . 119
4.9.2 The VPE table . . . 124
4.10 Virtual interrupts . . . 128
4.10.1 Virtual LPIs . . . 128
4.10.2 Virtual SPIs . . . 131
4.10.3 Virtual interrupt routing . . . 134
4.10.4 Virtual interrupt signaling . . . 135
4.10.5 VPE selection and configuration . . . 137
4.10.6 VPE residency . . . 138
4.10.7 VPE doorbells . . . 138
4.10.8 1ofN doorbells . . . 140
4.10.9 Save and restore of virtual interrupts . . . 140
4.11 IRS power management . . . 143
4.12 IRS memory access rules . . . 145
4.13 IRS support for MPAM . . . 148
4.14 IRS support for Memory Encryption Contexts . . . 150
4.15 IRS support for software error reporting . . . 151

**Chapter 5 Interrupt translation service (ITS)**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

5.1 ITS Domains . . . 157
5.1.1 Supporting Realm interrupts from Non-secure writes . . . 158
5.2 Operation . . . 160
5.2.1 Enabling and disabling the ITS . . . 160
5.2.2 Interrupt event types . . . 161
5.2.3 Software generated ITS events . . . 161
5.2.4 ITS synchronization requests . . . 162
5.3 Translation structures . . . 163
5.3.1 The Device Table (DT) . . . 164
5.3.2 The Interrupt Translation Table (ITT) . . . 166
5.4 ITS cache management . . . 169
5.4.1 ITS cache management for EventIDs . . . 170
5.4.2 ITS cache management for DeviceIDs . . . 171
5.5 ITS memory access rules . . . 173
5.6 ITS support for MPAM . . . 174
5.7 ITS support for Memory Encryption Conte . . . 176
5.8 ITS support for software error reporti . . . 177

**Chapter 6 Interrupt Wire Bridge (IWB)**
6.1 IWB wire control registers . . . 183
6.2 IWB support for multiple Interrupt ains . . . 186

**Chapter 7 GIC Performance Monito ng U t (PM )**
7.1 CoreSight PMU e nsions . . . 190
7.2 GIC PMU Overflow errup . . . 192
7.3 GIC PMU eve types . . . 193
7.4 Ordering d sy ronizat n requirements for the GIC PMU . . . 194
7.5 Event f ering . . . 195
7.6 IRS M events . . . 196
7.6.1 IRS MU events filtering . . . 198
7.7 TS PMU eve s . . . 200
7.7.1 ITS PMU vents filtering . . . 200

**Chapter 8 Sy tem i tructions**
8.1 System i tructions for the Current Interrupt Domain . . . 203
8. IC CDAFF, Interrupt Set Target in the Current Interrupt Domain . . . 204
8.1.2 GIC CDDI, Interrupt Deactivate in the Current Interrupt Domain . . . 206
8.1.3 GIC CDDIS, Interrupt Disable in the Current Interrupt Domain . . . 208
8.1.4 GIC CDEN, Interrupt Enable in the Current Interrupt Domain . . . 210
8.1.5 GIC CDEOI, Priority Drop in the Current Interrupt Domain . . . 212
8.1.6 GIC CDHM, Interrupt Handling mode state in the Current Interrupt Domain 214
8.1.7 GIC CDPEND, Interrupt Set/Clear Pending state in the Current Interrupt Domain . . . 216
8.1.8 GIC CDPRI, Interrupt Set priority in the Current Interrupt Domain . . . 218
8.1.9 GIC CDRCFG, Request Interrupt Configuration in the Current Interrupt Domain . . . 220
8.1.10 GICR CDIA, Interrupt Acknowledge in the Current Interrupt Domain . . . 222
8.1.11 GICR CDNMIA, Non-maskable Interrupt Acknowledge in the Current Interrupt Domain . . . 224
8.2 System instructions for the Virtual Interrupt Domain . . . 226
8.2.1 GIC VDAFF, Interrupt Set Target in the Virtual Interrupt Domain . . . 227
8.2.2 GIC VDDI, Interrupt Deactivate in the Virtual Interrupt Domain . . . 229
8.2.3 GIC VDDIS, Interrupt Disable in the Virtual Interrupt Domain . . . 231
8.2.4 GIC VDEN, Interrupt Enable in the Virtual Interrupt Domain . . . 233
8.2.5 GIC VDHM, Interrupt Handling mode in the Virtual Interrupt Domain . . . 235

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

8.2.6 GIC VDPEND, Interrupt Set/Clear Pending state in the Virtual Interrupt Domain . . . 237
8.2.7 GIC VDPRI, Interrupt Set priority in the Virtual Interrupt Domain . . . 239
8.2.8 GIC VDRCFG, Request Interrupt Configuration in the Virtual Interrupt Domain . . . 241
8.3 System instructions for the Logical Interrupt Domain . . . 243
8.3.1 GIC LDAFF, Interrupt Set Target in the Logical Interrupt Domain . . . 244
8.3.2 GIC LDDI, Interrupt Deactivate in the Logical Interrupt Domain . . . 246
8.3.3 GIC LDDIS, Interrupt Disable in the Logical Interrupt Domain . . . 248
8.3.4 GIC LDEN, Interrupt Enable in the Logical Interrupt Domain . . . 250
8.3.5 GIC LDHM, Interrupt Handling mode in the Logical Interrupt Domain . . . 252
8.3.6 GIC LDPEND, Interrupt Set/Clear Pending state in the Logical Interrupt Domain . . . 254
8.3.7 GIC LDPRI, Interrupt Set priority in the Logical Interrupt Domain . . . 256
8.3.8 GIC LDRCFG, Request Interrupt Configuration in the Logical Interrupt Domain . . . 258
8.4 GIC synchronization barrier instructions . . . 260
8.4.1 GSB SYS, GIC Synchronization Barrier System . . . 261
8.4.2 GSB ACK, GIC Synchronization Barrier Interrupt Acknowledge . . . 262

**Chapter 9 System registers**
9.1 Synchronization requirements for GICv5 System registers . . . 264
9.2 CPU interface registers . . . 265
9.2.1 ICC_APR_EL1, Interrupt Controller Physical Active Priorities Register (EL1) . . . 266
9.2.2 ICC_APR_EL3, Interrupt Controller Physical Active Priorities Register for EL3 . . . 269
9.2.3 ICC_CR0_EL1, Interrupt Controller Physical Control Register (EL1) . . . 271
9.2.4 ICC_CR0_EL3, Interrupt Controller Physical Control Register (EL3) . . . 276
9.2.5 ICC_DOMHPPIR_EL3, Interrupt Controller Domain Highest Priority Pending Interrupt Register . . . 279
9.2.6 ICC_HAPR_EL1, Interrupt Controller Physical Highest Active Priority Register . . . 282
9.2.7 ICC_HPPIR_EL1, Interrupt Controller Physical Highest Priority Pending Interrupt Register (EL1) . . . 284
9.2.8 ICC_HPPIR_EL3, Interrupt Controller Physical Highest Priority Pending Interrupt Register (EL3) . . . 286
9.2.9 ICC_IAFFIDR_EL1, Interrupt Controller PE Interrupt Affinity ID Register 288
9.2.10 ICC_ICSR_EL1, Interrupt Controller Interrupt Configuration and State Register . . . 289
9.2.11 ICC_IDR0_EL1, Interrupt Controller ID Register 0 . . . 293
9.2.12 ICC_PCR_EL1, Interrupt Controller Physical Interrupt Priority Control Register (EL1) . . . 295
9.2.13 ICC_PCR_EL3, Interrupt Controller Interrupt Priority Control Register (EL3) . . . 298
9.2.14 ID_AA64PFR2_EL1, AArch64 Processor Feature Register 2 . . . 300
9.3 Virtual CPU interface registers . . . 302
9.3.1 ICV_APR_EL1, Interrupt Controller Virtual Active Priorities Register . . . 303
9.3.2 ICV_CR0_EL1, Interrupt Controller EL1 Virtual Control Register . . . 306
9.3.3 ICV_HAPR_EL1, Interrupt Controller Virtual Highest Active Priority Register . . . 309
9.3.4 ICV_HPPIR_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register . . . 311
9.3.5 ICV_PCR_EL1, Interrupt Controller Virtual Interrupt Priority Control Register . . . 313

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

9.4 PPI registers . . . 316
9.4.1 ICC_PPI_CACTIVER<n>_EL1, Interrupt Controller Physical PPI Clear Active Registers, n = 0 - 1 . . . 317
9.4.2 ICC_PPI_CPENDR<n>_EL1, Interrupt Controller Physical PPI Clear Pending State Registers, n = 0 - 1 . . . 320
9.4.3 ICC_PPI_DOMAINR<n>_EL3, Interrupt Controller PPI Domain Registers, n = 0 - 3 . . . 323
9.4.4 ICC_PPI_ENABLER<n>_EL1, Interrupt Controller Physical PPI Enable Registers, n = 0 - 1 . . . 325
9.4.5 ICC_PPI_HMR<n>_EL1, Interrupt Controller Physical PPI Handling mode Registers, n = 0 - 1 . . . 328
9.4.6 ICC_PPI_PRIORITYR<n>_EL1, Interrupt Controller Physical PPI Priority Registers, n = 0 - 15 . . . 330
9.4.7 ICC_PPI_SACTIVER<n>_EL1, Interrupt Controller Physical PPI Set Active Registers, n = 0 - 1 . . . 332
9.4.8 ICC_PPI_SPENDR<n>_EL1, Interrupt Controller Physical PPI Set Pending State Registers, n = 0 - 1 . . . 335
9.5 Virtual PPI registers . . . 338
9.5.1 ICV_PPI_CACTIVER<n>_EL1, Interrupt Controller Virtual PPI Clear Active Registers, n = 0 - . . . 339
9.5.2 ICV_PPI_CPENDR<n>_EL1, Interrupt Controller Virtual PPI Clear Pending State Registers, n = 0 - 1 . . . 342
9.5.3 ICV_PPI_ENABLER<n>_EL1, Interrupt Controller Virtual PPI Clear Enable Registers, n = 0 . . . 345
9.5.4 ICV_PPI_HMR<n>_EL1, Interrupt Controller Virtual PPI Handling mode Register, n = 0 . . . 348
9.5.5 ICV_PPI_PRIORITYR<n>_EL1, Interrupt Controller Virtual PPI Priority Registers, n = 0 - 15 . . . 350
9.5.6 ICV_PPI_SACTIVER<n>_EL1, Interrupt Controller Virtual PPI Set Active Registers, n = 0 - 1 . . . 352
9.5.7 ICV_PPI_SPENDR<n>_EL1, Interrupt Controller Virtual PPI Set Pending State Registers, n = 0 - 1 . . . 355
9.6 Hypervisor control registers . . . 358
9.6.1 ICH_APR_EL2, Interrupt Controller Active Virtual Priorities Register . . . 359
9.6.2 ICH_CONTEXTR_EL2, Interrupt Controller Virtual Context Register . . . 361
9.6.3 ICH_HFGITR_EL2, Hypervisor GIC Fine-Grained Instruction Trap Register 365
9.6.4 ICH_HFGRTR_EL2, Hypervisor GIC Fine-Grained Read Trap Register 371
9.6.5 ICH_HFGWTR_EL2, Hypervisor GIC Fine-Grained Write Trap Register 378
9.6.6 ICH_HPPIR_EL2, Interrupt Controller Hypervisor Highest Priority Pending Interrupt Register . . . 383
9.6.7 ICH_PPI_ACTIVER<n>_EL2, Interrupt Controller Virtual Interrupt Active Registers, n = 0 - 1 . . . 385
9.6.8 ICH_PPI_DVIR<n>_EL2, Interrupt Controller PPI Direct-inject Virtual Interrupt Registers, n = 0 - 1 . . . 388
9.6.9 ICH_PPI_ENABLER<n>_EL2, Interrupt Controller Virtual Interrupt Enable Registers, n = 0 - 1 . . . 391
9.6.10 ICH_PPI_PENDR<n>_EL2, Interrupt Controller Virtual Interrupt Pending State Registers, n = 0 - 1 . . . 393
9.6.11 ICH_PPI_PRIORITYR<n>_EL2, Interrupt Controller Virtual Interrupt Priority Registers, n = 0 - 15 . . . 396
9.6.12 ICH_VCTLR_EL2, Interrupt Controller Virtual CPU interface Control Register . . . 398
9.6.13 ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register 401
9.6.14 Nested virtualization . . . 405
9.7 Legacy hypervisor control registers . . . 406

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

9.7.1 ICH_AP0R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3 . . . 407
9.7.2 ICH_AP1R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3 . . . 411
9.7.3 ICH_EISR_EL2, Interrupt Controller End of Interrupt Status Register . . 415
9.7.4 ICH_ELRSR_EL2, Interrupt Controller Empty List Register Status Register417
9.7.5 ICH_HCR_EL2, Interrupt Controller Hyp Control Register . . . 419
9.7.6 ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15 . . . 426
9.7.7 ICH_MISR_EL2, Interrupt Controller Maintenance Interrupt State Register432
9.7.8 ICH_VTR_EL2, Interrupt Controller VGIC Type Register . . . 436
9.7.9 Nested virtualization . . . 440
9.8 Legacy virtual CPU interface registers . . . 441
9.8.1 AArch64 Legacy virtual CPU interface registers . . . 441

**Chapter 10 Registers and memory maps**
10.1 Memory-mapped programmer's model . . . 444
10.2 IRS register frames . . . 447
10.2.1 IRS_CONFIG_FRAME . . . 447
10.2.2 IRS_SETLPI_FRAME . . . 544
10.3 ITS register frames . . . 547
10.3.1 ITS_CONFIG_FRAME . . . 547
10.3.2 ITS_TRANSLATE_FRAME . . . 597
10.4 IWB register frames . . . 601
10.4.1 IWB_CONFIG_FRAME . . . 601
10.5 GIC PMU register frame . . . 619
10.5.1 GIC_PMU_FRAME . . . 619
10.6 CoreSight identification registers . . . 637

**Chapter 11 Data structures**
11.1 ITS Data Structures . . . 640
11.1.1 L1_DTE, Level 1 device table entry . . . 641
11.1.2 L2_DTE, Level 2 device table entry . . . 642
11.1.3 L1_ITTE, Level 1 interrupt translation table entry . . . 645
11.1.4 L2_ITTE, Level 2 interrupt translation table entry . . . 646
11.2 IRS Data Structures . . . 648
11.2.1 L1_VMTE, Level 1 VM table entry . . . 649
11.2.2 L2_VMTE, Level 2 VM table entry . . . 650
11.2.3 L1_ISTE, Level 1 interrupt state table entry . . . 657
11.2.4 L2_ISTE, Level 2 interrupt state table entry . . . 658
11.2.5 VPETE, VPE table entry . . . 660
11.2.6 VM_DESC, VM descriptor . . . 661
11.2.7 VPE_DESC, VPE descriptor . . . 662

**Part A GICv5 Stream Protocol interface**

**Chapter A1 GICv5 Stream Protocol overview**

**Chapter A2 AMBA AXI5-Stream Transport Layer**
A2.1 Signals . . . 668
A2.2 Channel identification . . . 670
A2.3 Link status . . . 672

**Chapter A3 Common behaviors**

**Chapter A4 Interrupt Handling channel**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A4.1 Command summary . . . 675
A4.2 Outstanding commands . . . 677
A4.3 Connection management . . . 679
A4.4 Managing the resident VPE . . . 682
A4.4.1 Interrupt Handling Channel behaviors when there is a resident VPE . . 683
A4.4.2 Interrupt Handling Channel behaviors when there is no resident VPE . 683
A4.5 Forwarding, recalling, and releasing interrupts . . . 684
A4.6 INTID configuration . . . 686
A4.7 IRS and CPU interface capabilities . . . 687

**Chapter A5 Interrupt Signaling channel**
A5.1 Command summary . . . 689
A5.2 Outstanding commands . . . 690
A5.3 Signaling interrupts to the IRS . . . 691
A5.4 Connection management . . . 693

**Chapter A6 Alphabetical list of commands**
A6.1 Interrupt Handling channel . . . 696
A6.1.1 Activate, Activate command (CPUIF -> IRS) . . . 697
A6.1.2 ActivateAck, Activate Acknowledge command (IRS -> CPUIF) . . . 698
A6.1.3 Deactivate, Deactivate interrupt command (CPUIF -> IRS) . . . 699
A6.1.4 DeactivateAck, Deactivate Interrupt Acknowledge command (IRS -> CPUIF) . . . 701
A6.1.5 DownstreamControl, Downstream Control (IRS -> CPUIF) . . . 702
A6.1.6 DownstreamControlAck, Downstream Control Acknowledge command (CPUIF -> IRS) . . . 703
A6.1.7 Forward, Forward command (IRS -> CPUIF) . . . 704
A6.1.8 Recall, Recall command (IRS -> CPUIF) . . . 706
A6.1.9 Release, Release command (CPUIF -> IRS) . . . 707
A6.1.10 RequestConfig, Request Interrupt Configuration command (CPUIF -> IRS) 709
A6.1.11 RequestConfigAck, Request Interrupt Configuration Acknowledge command (IRS -> CPUIF) . . . 711
A6.1.12 SetAck, Set interrupt configuration acknowledge command (IRS -> CPUIF) 713
A6.1.13 SetEnabled, Set interrupt Enabled command (CPUIF -> IRS) . . . 714
A6.1.14 SetHandling, Set Interrupt Handling Mode command (CPUIF -> IRS) . . . 716
A6.1.15 SetPending, Set interrupt Pending command (CPUIF -> IRS) . . . 718
A6.1.16 SetPriority, Set Interrupt Priority command (CPUIF -> IRS) . . . 720
A6.1.17 SetResident, Set Resident command (CPUIF -> IRS) . . . 722
A6.1.18 SetResidentAck, Set Resident acknowledge command (IRS -> CPUIF) 724
A6.1.19 SetTarget, Set Interrupt Target command (CPUIF -> IRS) . . . 725
A6.1.20 Sync, synchronizes previously sent configuration changes (CPUIF -> IRS) 727
A6.1.21 SyncAck, synchronizes previously sent configuration changes (IRS -> CPUIF) . . . 728
A6.1.22 UpstreamControl, Upstream Control (CPUIF -> IRS) . . . 729
A6.1.23 UpstreamControlAck, Upstream Control Acknowledge command (IRS -> CPUIF) . . . 730
A6.1.24 WakeRequest, Wake request (IRS -> CPUIF) . . . 731
A6.2 Interrupt Signaling channel . . . 732
A6.2.1 INT, Interrupt command (Interrupt Source -> IRS) . . . 733
A6.2.2 Flush, Flush command (IRS -> Interrupt Source) . . . 734
A6.2.3 FlushAck, Flush acknowledge command (Interrupt Source -> IRS) . . . 735
A6.2.4 Quiesce, Quiesce command (Interrupt Source -> IRS) . . . 736
A6.2.5 QuiesceAck, Quiesce acknowledge command (IRS -> Interrupt Source) 737
A6.2.6 Resample, Resample request command (IRS -> Interrupt Source) . . . 738

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A6.2.7 ResampleAck, Resample request acknowledge command (Interrupt Source -> IRS) . . . 739
A6.2.8 Reset, Reset command (Interrupt Source -> IRS) . . . 740
A6.2.9 ResetAck, Reset acknowledge command (IRS -> Interrupt Source) . . . 741

**Chapter A7 Example sequences**
A7.1 Bringing the Interrupt Handling channel online and taking it offline . . . 743
A7.2 Simple interrupt life-cycle . . . 748
A7.3 Replacing the candidate HPPI for an Interrupt Domain, or resident VPE . . . 749
A7.4 Making a VPE resident . . . 752
A7.5 Interrupt configuration . . . 755
A7.6 Sending IPIs . . . 756
A7.7 1 of N interrupts . . . 757

**Part B Litmus tests**

**Chapter B1 Interrupt ordering litmus tests**
B1.1 Interrupt litmus test assumptions . . . 760
B1.2 Atomicity of interrupt updates to GIC system instructions . . . 761
B1.2.1 Notes . . . 761
B1.2.2 Litmus test . . . 761
B1.3 Multiple updates of the same interrupt configuration . . . 762
B1.3.1 Notes . . . 762
B1.3.2 Litmus test with two configuration updates . . . 762
B1.3.3 Litmus test with an interrupt disable and an interrupt deactivate . . . 762
B1.3.4 Litmus test with an interrupt deactivate and an interrupt disable . . . 763
B1.4 Reading back interrupt write on a single PE . . . 764
B1.4.1 Notes . . . 764
B1.4.2 Litmus test with a configuration update . . . 764
B1.4.3 Litmus test with deactivate . . . 764
B1.4.4 Litmus test with acknowledgement . . . 765
B1.4.5 Litmus test with two configuration updates . . . 765
B1.5 Reading interrupt configurations and subsequent updates . . . 766
B1.5.1 Notes . . . 766
B1.5.2 Litmus test with update to priority . . . 766
B1.5.3 Litmus test with deactivate . . . 766
B1.5.4 Litmus test with acknowledge . . . 767
B1.6 Configuration and acknowledgement . . . 768
B1.6.1 Notes . . . 768
B1.6.2 Litmus test using disable without explicit synchronization . . . 768
B1.6.3 Litmus test using disable with explicit synchronization . . . 768
B1.6.4 Litmus test with deactivate . . . 769
B1.6.5 Litmus test using priority without explicit synchronization . . . 769
B1.6.6 Litmus test using priority with explicit synchronization. . . . 770
B1.7 Acknowledge followed by interrupt changes . . . 771
B1.7.1 Notes . . . 771
B1.7.2 Litmus test with deactivate . . . 771
B1.7.3 Litmus test with make pending . . . 771
B1.8 Multiple updates with interleaved read . . . 773
B1.8.1 Notes . . . 773
B1.8.2 Litmus test . . . 773
B1.9 Configuration write and IRQ unmask in PSTATE . . . 774
B1.9.1 Notes . . . 774
B1.9.2 Litmus test . . . 774

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B1.9.3 Litmus test with initially masked IRQs . . . 774
B1.9.4 Litmus test with disable of a PPI . . . 775
B1.10 Configuration write and exception status on a single PE . . . 776
B1.10.1 Notes . . . 776
B1.10.2 Litmus test without wait for IRQ exception to be signaled . . . 776
B1.10.3 Litmus test with wait for IRQ exception to be signaled . . . 776
B1.11 IPI and acknowledgement . . . 778
B1.11.1 Notes . . . 778
B1.11.2 Litmus test without explicit synchronization . . . 778
B1.11.3 Litmus test with explicit synchronization . . . 778
B1.12 Observing multiple writes on a different PE . . . 780
B1.12.1 Notes . . . 780
B1.12.2 Litmus test . . . 780
B1.13 Read of the configuration of an interrupt . . . 781
B1.13.1 Notes . . . 781
B1.13.2 Litmus test without ISB . . . 781
B1.13.3 Litmus test with ISB . . . 781
B1.14 Multiple reads of the same config . . . 783
B1.14.1 Notes . . . 783
B1.14.2 Litmus test with ISBs . . . 783
B1.15 GIC coherence order . . . 784
B1.15.1 Notes . . . 784
B1.15.2 Litmus test . . . 784
B1.16 Independent reads of independent writes . . . 785
B1.16.1 Notes . . . 785
B1.16.2 Litmus test . . . 785
B1.17 Message passing via a flag in memory . . . 786
B1.17.1 Notes . . . 786
B1.17.2 Litmus test . . . 786
B1.18 Message passing via interrupt priority configuration . . . 787
B1.18.1 Notes . . . 787
B1.18.2 Litmus test . . . 787
B1.19 Message passing with an LPI and a device read . . . 788
B1.19.1 Notes . . . 788
B1.19.2 Litmus test . . . 788
B1.19.3 Litmus test with address dependency . . . 788
B1.20 Message passing with an LPI and a GSB . . . 790
B1.20.1 Notes . . . 790
B1.20.2 Litmus test . . . 790
B1.21 Message passing with an IPI and a GSB . . . 791
B1.21.1 Notes . . . 791
B1.21.2 Litmus test . . . 791
B1.22 Message passing using deactivate . . . 792
B1.22.1 Notes . . . 792
B1.22.2 Litmus test with explicit synchronization . . . 792
B1.22.3 Litmus test with address dependency . . . 793
B1.22.4 Litmus test with control dependency . . . 793
B1.22.5 Litmus test without explicit synchronization . . . 794
B1.22.6 Litmus test with a DSB but without a GSB . . . 795
B1.22.7 Litmus test without a DSB but with a GSB . . . 796
B1.23 IPI edge merging and message passing . . . 797
B1.23.1 Notes . . . 797
B1.23.2 Litmus test . . . 797
B1.24 Device edge merging with GSB ACK . . . 798
B1.24.1 Notes . . . 798

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B1.24.2 Litmus test . . . 798
B1.25 Configuration read and interrupt acknowledge . . . 800
B1.25.1 Notes . . . 800
B1.25.2 Single-threaded litmus test . . . 800
B1.25.3 Multi-threaded litmus test . . . 800
B1.26 Atomicity of interrupt acknowledge . . . 802
B1.26.1 Notes . . . 802
B1.26.2 Litmus test . . . 802
B1.27 Atomicity of interrupt disable and acknowledge . . . 803
B1.27.1 Notes . . . 803
B1.27.2 Litmus test . . . 803
B1.28 Interrupt handler completion . . . 804
B1.28.1 Notes . . . 804
B1.28.2 Litmus test . . . 804
B1.29 Configuration update while disabled . . . 805
B1.29.1 Notes . . . 805
B1.29.2 Litmus test . . . 805
B1.30 Atomicity of interrupt acknowledge and retarget . . . 806
B1.30.1 Notes . . . 806
B1.30.2 Litmus test . . . 806
B1.31 1ofN interrupt acknowledge . . . 807
B1.31.1 Notes . . . 807
B1.31.2 Litmus test . . . 807
B1.32 Retargeting interrupts without synchronization . . . 808
B1.32.1 Notes . . . 808
B1.32.2 Litmus test . . . 808
B1.33 Reading interrupt configuration and exception status . . . 809
B1.33.1 Notes . . . 809
B1.33.2 Litmus test with ISB before reading IRQ pending status . . . 809
B1.33.3 Litmus test with ISB after reading IRQ pending status . . . 809
B1.34 Reading interrupt configuration and IRQ unmask in PSTATE . . . 811
B1.34.1 Notes . . . 811
B1.34.2 Explanation . . . 811
B1.34.3 Litmus test with ISB before unmasking IRQ in PSTATE . . . 811
B1.34.4 Litmus test with ISB after unmasking IRQ in PSTATE . . . 811
B1.35 PPI activate and System register read . . . 812
B1.35.1 Notes . . . 812
B1.35.2 Litmus test without ISB . . . 812
B1.35.3 Litmus test with GSB . . . 812
B1.35.4 Litmus test with ISB . . . 813
B1.36 PPI disable and acknowledge . . . 814
B1.36.1 Notes . . . 814
B1.36.2 Litmus test with ISB . . . 814
B1.37 PPI acknowledgement . . . 815
B1.37.1 Notes . . . 815
B1.37.2 Litmus test . . . 815
B1.37.3 Litmus test . . . 815
B1.38 Write after changing resident VM . . . 817
B1.38.1 Notes . . . 817
B1.38.2 Litmus test . . . 817
B1.39 Write before changing resident VM . . . 818
B1.39.1 Notes . . . 818
B1.39.2 Litmus test . . . 818
B1.40 Completion of GIC and GICR instructions in finite time . . . 819
B1.40.1 Notes . . . 819

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B1.40.2 Litmus test with a priority update observed by a configuration read . . . 819
B1.40.3 Litmus test with an interrupt acknowledge observed by a configuration read819
B1.40.4 Litmus test with an interrupt becoming pending and then acknowledged 820
B1.41 Effects of disabling a PPI source on the PPI Pending state . . . . . 821
B1.41.1 Notes . . . . . 821
B1.41.2 Litmus test with disable of the timer state . . . . . 821
B1.42 Priority mask updates and interrupt pending state and acknowledgement . . . 822
B1.42.1 Notes . . . . . 822
B1.42.2 Litmus test masking using the priority mask without synchronization . . 822
B1.42.3 Litmus test unmasking using the priority mask without synchronization . 822
B1.42.4 Litmus test masking using the priority mask and acknowledging without synchronization . . . . . 822
B1.42.5 Litmus test unmasking using the priority mask and acknowledging without synchronization . . . . . 823

**Part C Model**

**Chapter C1 Operational model**

**Glossary**

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Preface

## Conventions

### Typographical conventions

The typographical conventions are:

*italic*

Introduces special terminology, and denotes citations.

**bold**

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example https://developer.arm.com

### Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

### Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

### Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

## Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

## Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: $L_{iiiii}$

- $L$ is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

## Content item classes

### Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behavior.

A Declaration is rendered with the label *D*.

### Rule

A Rule is a statement that describes the behavior of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

A Rule is rendered with the label *R*.

### Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

### Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

### Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

### Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

### Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

## Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Arm® A-profile Architecture Reference Manual, for A-profile architecture*. (ARM DDI 0487) Arm Ltd.

[2] *Arm® Base System Architecture 1.0C*. (ARM DEN 0094C) Arm Ltd.

[3] *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4*. (ARM IHI 0069) Arm Ltd.

[4] *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3*. (ARM IHI 0070) Arm Ltd.

[5] *Arm® Realm Management Extension (RME) System Architecture*. (ARM DEN 0029) Arm Ltd.

[6] *AMBA® AXI Protocol Specification*. (ARM IHI 0022) Arm Ltd.

[7] *Arm® Power State Coordination Interface*. (ARM DEN 0022) Arm Ltd.

[8] *PCI Express® Base Specification Revision 6.0*. PCI-SIG.

[9] *VIRTIO (Virtual I/O) Specification*. See https://github.com/oasis-tcs/virtio-spec

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

[10] *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*. (ARM IHI 0091) Arm Ltd.

[11] *Arm® CoreSight™ Architecture Specification*. (ARM IHI 0029) Arm Ltd.

[12] *AMBA® AXI-Stream Protocol Specification*. (ARM IHI 0051) Arm Ltd.

[13] *Arm® Specification Language Reference Manual*. (ARM DDI 0612) Arm Ltd.

[14] *Arm® Reliability, Availability, and Serviceability (RAS) System Architecture for A-profile architecture*. (ARM IHI 0100) Arm Ltd.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Feedback

Arm welcomes feedback on its documentation.

## Feedback on this book

If you have any comments or suggestions for additions and improvements create a ticket at https://support.developer.arm.com. As part of the ticket include:

- The title (Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5).
- The number (ARM-AES-0070 00eac0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

---

**Note**

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

---

## Inclusive terminology commitment

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

Previous issues of this document included language that can be offensive. We have replaced this language. To report offensive language in this document, email terms@arm.com.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 1
# **Introduction**

RETIRED

$I_{SBNXT}$ The Generic Interrupt Controller version 5 (GICv5) architecture defines requirements for interrupt sources and a common programming interface for translating, configuring, and handling interrupts.

$I_{JCSXY}$ Sources of interrupts can be peripherals (including PE functions, system components, and the GIC itself), and software running on PEs.

$I_{JCVZT}$ The GICv5 CPU interface provides a common programming interface to manage and handle physical and virtual interrupts.

Virtual interrupts are handled using the same programming interface as physical interrupts, providing binary compatible virtualization support.

The GICv5 CPU interface also supports backwards compatibility for GICv3 virtual machines by providing EL2 controls as well as the GICv3 virtual CPU interface and GICv3 hypervisor control interface. Backward compatibility support in the GICv5 CPU interface is an optional feature.

$I_{NHXNK}$ To support virtualization, the GICv5 architecture provides mechanisms for managing interrupts for Virtual Machines (VMs) and virtual PEs (VPEs). The architecture supports multiple VMs, and each VM can contain one or more VPEs.

Software can configure when a VPE becomes resident on a PE through a System register interface. It can also request doorbell notification interrupts when making a VPE non-resident at a later time.

The GIC architecture does not define or mandate how a VM boundary is managed by software.

The following are examples of VMs that the GICv5 architecture supports:

- A VM running in Non-secure state at EL1 and EL0, managed by a hypervisor running in Non-secure state at EL2.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- A Realm running in Realm state at EL1 and EL0, and is managed by the Realm Management Monitor (RMM) running in Realm state at EL2.
- A Secure partition running in Secure state at EL1 and EL0, and is managed by the Secure Partition Manager (SPM) running in Secure state at EL2.

A VPE represents an execution context within a VM.

$D_{QJTJG}$ A VM is identified by a VM ID.

A VPE is identified by a VPE ID in the context of a VM.

$I_{NBJJF}$ VMs are currently specified using a 16-bit identifier. The architecture reserves space to expand to additional bits of VM ID across all registers and data structures in a future version of the architecture. Due to the need to support a VPE doorbell in the LPI INTID space for each VPE in each VM, expanding the number of VMs beyond 24 bits may also require expanding the number of INTID.IDs supported.

$D_{BJBPZ}$ A GICv5 CPU interface is connected to the *Interrupt Routing Infrastructure (IRI)*.

The IRI uses either the GICv5 system architecture as described in Chapter 3 *GICv5 system architecture*, or is IMPLEMENTATION DEFINED.

If the IRI is IMPLEMENTATION DEFINED, it must meet the requirements for PPIs and LPIs specified in Chapter 2 *PE architecture*, including the requirements specified in 2.12 *Interrupt ordering model and synchronization requirements*.

$D_{GMPXJ}$ In the context of this specification, a GICv5 *system* comprises one or more PEs with support for the GICv5 architecture, each connected to the IRI.

This specification expects that the same OS or hypervisor runs all the PEs in a GICv5 system.

The word system can be used in different contexts to describe a different set of components, for example to include PEs that do not implement support for the GICv5 architecture.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter 2
# PE architecture

RETIRED

## 2.1 Architecture features and extensions

$R_{PNBFY}$ GICv5 defines a new Armv9-A architecture feature, GICv5 CPU interface Extension, represented by FEAT_GCIE.

This feature is supported in AArch64 only.

$I_{GPWSJ}$ Support for FEAT_GCIE is reported in ID_AA64PFR2_EL1.GCIE.

$R_{JMHXV}$ FEAT_GCIE is optional from Armv9.3-A[1].

$I_{SHDMS}$ FEAT_NMI is mandatory from Armv8.8-A and Armv9.3-A. This means that when a PE implements FEAT_GCIE, it also implements FEAT_NMI.

$R_{KTBJL}$ A PE that implements FEAT_GCIE does not implement FEAT_GICv3.

$I_{KNDMT}$ When a PE implements FEAT_GCIE, ID_AA64PFR0_EL1.GIC is 0.

$R_{BQYTD}$ GICv5 defines an optional Armv9-A architecture feature, GICv5 legacy VPE support, represented by FEAT_GCIE_LEGACY.

$I_{BKBQV}$ FEAT_GCIE_LEGACY is implemented by a PE if ICC_IDR0_EL1.GCIE_LEGACY is >= 1. Otherwise, FEAT_GCIE_LEGACY is not implemented.

$I_{CXTVF}$ FEAT_GCIE_LEGACY provides a virtual CPU interface compatible with GICv3.3, including:

- GICv3.3 ICV registers.
- GICv3.3 ICH registers, with some GICv5 updates.
- GICv3.3 support for virtual Non-maskable interrupts.
- GICv3.1 extended INTID range.
- GICv3 maintenance interrupt.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- GICv3 support for separate trapping of ICV_DIR_EL1.

See 2.10.7 *Legacy virtual CPU interface* for more information.

$R_{QCWBY}$ A PE that implements FEAT_GCIE_LEGACY also implements FEAT_GCIE and EL2.

$R_{VVMRJ}$ When a PE implements FEAT_GCIE, SCR_EL3.IRQ is RES0.

$R_{JMXDP}$ When FEAT_GCIE and FEAT_EL3 are implemented on a PE, SCR_EL3.FIQ is RES1.

See also:

- 2.7 *Interrupt Prioritization*
- 2.9 *The physical CPU interface*
- 2.9.3 *Preemptive interrupts*
- 3.1 *Interrupt Domains*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 2.2 The GICv5 CPU interface

I_GNFVQ When FEAT_GCIE is implemented, the PE implements the GICv5 CPU interface. The GICv5 CPU interface is responsible for:

- Managing configuration and state of *Private Peripheral Interrupts* (PPIs).
- Masking and signaling of interrupts.
- Providing an interface for software to handle interrupts.
- Providing an interface for software to configure *Shared Peripheral Interrupts* (SPIs) and *Logical Peripheral Interrupts* (LPIs).
- If EL2 is implemented, the CPU interface supports both virtual interrupt and physical interrupts.

A PE with an integrated GICv5 CPU interface is illustrated in Figure 2.1.

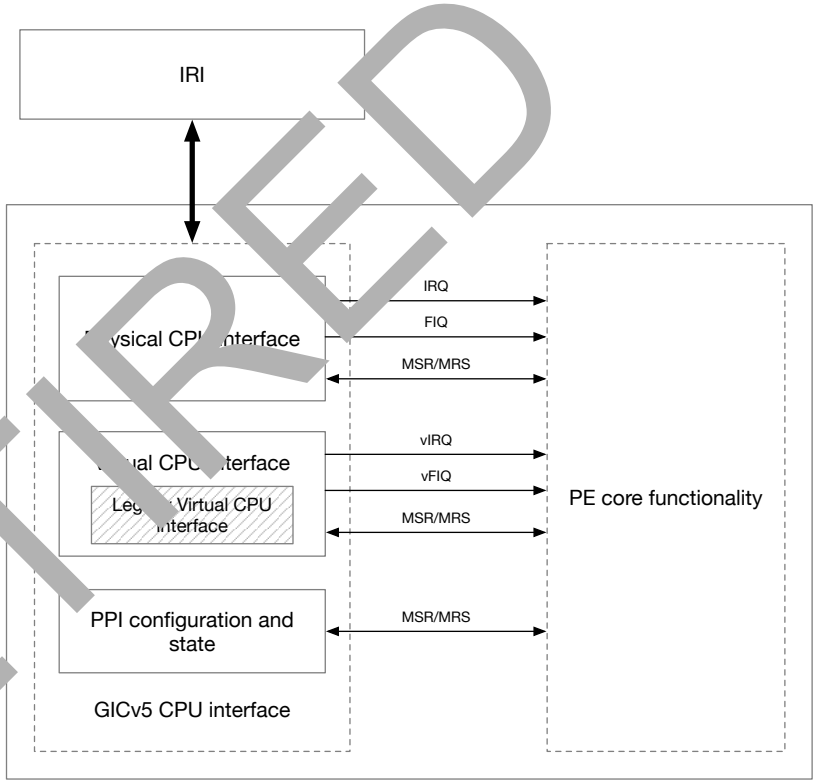



**Figure 2.1: PE with an integrated GICv5 CPU interface.**

RETIRED

**Note**

vFIQ is only signaled by the Legacy virtual CPU interface.

I_CMPTD The GICv5 CPU interface defines when an interrupt is signaled to the PE.

The PE architecture supports the IRQ, FIQ, vIRQ, and vFIQ interrupt signals. These interrupts signals can have Superpriority as an additional attribute.

The PE architecture defines additional rules detailing whether an interrupt signal results in an exception being taken. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information.

I_PYKTM The physical CPU interface is always active and determines whether the IRQ and FIQ interrupts are signaled. See *2.9 The physical CPU interface* for more information.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

When executing at Exception levels below EL2, HCR_EL2.IMO is 1, and Legacy operation is not enabled, the virtual CPU interface determines whether the vIRQ interrupt is signaled. See 2.10 *The virtual CPU interface* for more information.

When executing at Exception levels below EL2, HCR_EL2.IMO is 1, and Legacy operation is enabled, the Legacy virtual CPU interface determines whether the vIRQ and vFIQ interrupts are signaled. See 2.10.7 *Legacy virtual CPU interface* for more information.

The GICv5 CPU interface never signals the vIRQ or vFIQ interrupts when executing at EL2 or EL3.

$D_{MJHKK}$ The GICv5 CPU interface connects to the IRI that manages SPIs and LPIs. The IRI *presents* a candidate HPPI for each Interrupt Domain to the CPU interface and the CPU interface determines if the candidate HPPI is the HPPI.

$R_{MCVBY}$ The mechanism used by the IRI to present a candidate HPPI is IMPLEMENTATION DEFINED.

$I_{BTNJQ}$ Arm recommends that an implementation uses the GICv5 system archite See Chapter 3 *GICv5 system architecture* for more information.

Arm recommends that an implementation uses the GICv5 Stream Pr col for comm ication between PEs and the IRI. See Chapter A1 *GICv5 Stream Protocol overview* for m re info ation.

$R_{WKGBT}$ Every PE is assigned a unique PE interrupt Affinity ID in th system.

$I_{XDCZN}$ The interrupt Affinity ID value for each PE is reported in C_IA IDR_EL1.IAFFID.

$R_{GXVVJ}$ The PE interrupt Affinity ID is the same across all P ical In rupt Do ins.

$R_{PXKWK}$ Support for bypassing the GICv5 CPU interfac o signal nterrupts e PE is IMPLEMENTATION DEFINED. The behavior of System instruction and System re ters, if GICv5 CPU interface is bypassed, is IMPLEMENTATION DEFINED.

$R_{TYFBF}$ All GIC System instructions define in thi pecific on are unchanged in Debug state.

$I_{GDPVW}$ Instructions that move System Special-purp e registers to or from a general-purpose register are unchanged in Debug state. This includ in tructions that m ve System registers defined in this specification. See 'A64 instructions that are unchanged in De g state' in [1] for more information.

See also:

- 2.8 *Interr handli*

### 2.2.1 IRI link ma ment

$I_{JSRRY}$ ICC R0_EL LINK rep ts whether the communication link between the CPU interface and the IRI is con ted. S request the link to be connected or disconnected by writing to ICC_CR0_ELx.LINK. ICC_C ELx.LINK_IDLE reports whether an attempt to connect or disconnect the communication link between the CPU in face and the IRI is in progress.

$I_{YYQLX}$ If EL3 is imp emented, the communication link between the CPU interface and the IRI is managed using ICC_CR0_EL3.{LINK, LINK_IDLE} and the corresponding fields in ICC_CR0_EL1 are read-only aliases of the fields in ICC_CR0_EL3. Otherwise, the communication link between the CPU interface and the IRI is managed using ICC_CR0_EL1.{LINK, LINK_IDLE}.

$R_{VXKRR}$ When ICC_CR0_ELx.LINK is 0, all of the following are true:

- All virtual and physical LPIs and SPIs are unreachable.
- There is no resident VPE.
- A write that sets ICH_CONTEXTR_EL2.V to 1 results in ICH_CONTEXTR_EL2.F being set to 1.

Otherwise, when ICC_CR0_ELx.LINK is 1, LPIs and SPIs are reachable as defined in 2.6 *GIC System instructions* and management of the resident VPE behaves as described in 2.10.5 *Selecting the resident VPE*.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{LKMNG}$ When ICC_CR0_ELx.LINK_IDLE is 0, all of the following are true:

- It is CONSTRAINED UNPREDICTABLE whether virtual and physical LPIs and SPIs are unreachable.
- If ICH_CONTEXTR_EL2.V is 1, it is CONSTRAINED UNPREDICTABLE whether the PE behaves as if there is no resident VPE.
- It is CONSTRAINED UNPREDICTABLE whether a write that sets ICH_CONTEXTR_EL2.V to 1 results in ICH_CONTEXTR_EL2.F being set to 1.

$R_{NLCZC}$ When there are Interrupt Effects that are not yet complete from GIC System instructions in program order before a write that sets ICC_CR0_ELx.LINK to 0, it is CONSTRAINED UNPREDICTABLE whether those Interrupt Effects complete.

$S_{FHHYP}$ After writing ICC_CR0_ELx.LINK to 1 and reading ICC_CR0_ELx.LINK_IDLE as 1, an ISB is required to guarantee that subsequent writes to GIC System registers or execution of GIC System instructions see the link as connected.

Arm recommends that before setting ICC_CR0_ELx.LINK to 0, software performs all of the following actions:

- Issues a `GSB SYS` instruction after the last GIC System instruction.
- Ensures that there is no resident VPE.

$I_{BWPQB}$ If ICC_CR0_ELx.LINK_IDLE is 0, it will become 1 in a finite time. This means that software may poll ICC_CR0_ELx.LINK_IDLE until it becomes idle, for example during early software initialization sequences.

$R_{BPKBY}$ The reset value of ICC_CR0_ELx.{LINK, LINK_IDLE} is IMPLEMENTATION DEFINED to one of the following permitted combinations:

| LINK | LINK_IDLE | Meaning |
|---|---|---|
| 0 | 1 | The CPU interface is disconnected from the IRI at reset. |
| 1 | 1 | The CPU interface is connected to the IRI at reset. |
| 1 | 0 | The CPU interface is in the process of connecting to the IRI at reset. |

ICC_CR0_ELx.{LINK, LINK_IDLE} is not permitted to reset to {0, 0}.

$S_{FQXQW}$ The IRS or other system components might require initialization post reset before the link can be connected. If such initialization is required, software must not attempt to connect the link before the required initialization is complete.

$S_{YQQJL}$ When software writes to update the value of ICC_CR0_ELx.LINK, following the write, software is expected to poll ICC_CR0_ELx.LINK_IDLE until it returns 1.

$I_{RXKYV}$ Arm strongly recommends that the link is established before leaving EL3 or running any generic code.

$R_{MBRGG}$ When ICC_CR0_ELx.LINK_IDLE is 1, ICC_CR0_ELx.LINK is only updated as a result of a direct System register write or as a result of a CPU reset.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.3 Interrupt Domains

$D_{HJYDR}$ An *Interrupt Domain* provides isolation of interrupt configuration and handling by its association to a Security state.

$D_{XQLTP}$ The architecture defines the following *Physical Interrupt Domains*, each corresponding to an implemented Security state and Exception level:

- The Non-secure Interrupt Domain
- The Realm Interrupt Domain
- The Secure Interrupt Domain
- The EL3 Interrupt Domain

$D_{TTZCX}$ When an interrupt belonging to a *Physical Interrupt Domain* is signaled, it is signaled as a physical interrupt.

See 2.9 *The physical CPU interface* and 'Asynchronous exception type…in [1] fo…ore information.

$D_{HXMSH}$ The architecture defines the term *Non-EL3 Interrupt Domain* as any P…ical Interrupt …omain apart from the EL3 Interrupt Domain.

$I_{NHCWZ}$ The Non-secure, Secure, and Realm Interrupt Domains are …n-EL3 Interru… Do…ains.

$I_{WBZXZ}$ Physical PPIs have a per-PE namespace that is shared ac… the P…sical Interrupt Domains.

$I_{VWGBP}$ Physical SPIs have a single shared namespace acro…Es and …Physic…nterrupt Domains.

$I_{NFQYW}$ Physical LPIs have a separate namespace for e…n Physi…l Interru…omain that is shared across PEs.

$R_{ZMRKX}$ The implemented Physical Interrupt Domains …epend… …mented Security states as follows:

| …nplem…d Sec…y states | Implemented Physical Interrupt Domains |
|---|---|
| …on-secure | Non-secure |
| Sec… | Secure |
| Non-sec…re, Secure | Non-secure, Secure, EL3 |
| Non-secure, Realm, Root | Non-secure, Realm, EL3 |
| Non-secure, Realm, Secure, Root | Non-secure, Realm, Secure, EL3 |

$R_{ZFCXM}$ The P…sica…Interru…omain associated with a Security state and Exception level is as follows:

| Exception level | Security state | Physical Interrupt Domain |
|---|---|---|
| EL3 | Secure or Root[a] | EL3 |
| EL2/1/0 | Secure | Secure |
| EL2/1/0 | Non-secure | Non-secure |
| EL2/1/0 | Realm | Realm |

a. The Security state of EL3 depends on whether FEAT_RME is implemented. However, EL3 always uses the EL3 Interrupt Domain.

$D_{XKSVB}$ The architecture defines the *Current Physical Interrupt Domain* of a PE as the Physical Interrupt Domain corresponding to the current Exception level and Security state of that PE.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{FDPCN}$ For example, if a PE is running in EL3 on a system with the Secure and Non-secure Security states, the *Current Physical Interrupt Domain* is the EL3 Interrupt Domain.

$D_{QWVZJ}$ When EL2 is implemented, the *Virtual Interrupt Domain* defines the namespace for the virtual interrupts on a PE.

When EL2 is not implemented, there is no Virtual Interrupt Domain.

$D_{QHMVR}$ The interrupts managed by the IRI for the Virtual Interrupt Domain are selected based on the *resident VPE*.

The VM that the VPE belongs to is referred to as the *resident VM*.

$I_{VHGTD}$ The resident VPE and resident VM are programmed using ICH_CONTEXTR_EL2.

See 2.10.5 *Selecting the resident VPE* for more information.

$D_{ZWDFY}$ When an interrupt belonging to the *Virtual Interrupt Domain* is signaled, it is signaled as a virtual interrupt.

See 2.10 *The virtual CPU interface* and 'Asynchronous exception types' [1] for more information.

$D_{NSTKG}$ The *Current Interrupt Domain* is one of the following:

- The Virtual Interrupt Domain, when all of the following are true:
  - The current Exception level is EL1.
  - HCR_EL2.IMO is 1.
  - ICH_VCTLR_EL2.V3 is 0.
- Otherwise, the Current Physical Interrupt Domain.

See 2.6 *GIC System instructions* for more information about the use of the Current Interrupt Domain for GIC System instructions.

---

**Note**

The Current Interrupt Domain is not defined when ICH_VCTLR_EL2.V3 is 1 and executing at Exception levels below EL2. See 2.10.7 *Legacy virtual CPU interface* for more information.

---

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 2.4 Interrupt types and identifiers

$D_{SHNLW}$ FEAT_GCIE supports the following types of interrupts:

- Private Peripheral Interrupts (PPIs).
- Logical Peripheral Interrupts (LPIs).
- Shared Peripheral Interrupts (SPIs).

$D_{QDWHN}$ An *interrupt* is identified by an INTID which comprises the following:

- **INTID.TYPE** The type of an interrupt is PPI, LPI, or SPIs.
- **INTID.ID** The numeric ID of an interrupt.

Each interrupt comprises the following states:

- **Pending** An interrupt is either Pending or Idle. An interrupt is only signaled and acknowledged when it is Pending.
- **Active** An interrupt is either Active or Inactive. An interrupt is only signaled and acknowledged when it is Inactive.

Each interrupt comprises the following configuration values:

- **Enabled** An interrupt is either Enabled or Disabled. An interrupt is only signaled and acknowledged when it is Enabled.
- **Priority** The priority value of an interrupt. The priority values determine the order in which interrupts are acknowledged and allow masking interrupts of lower priority.
- **Handling mode** The Handling mode of an interrupt is either Edge or Level. When the Handling mode is Edge, the Pending state becomes Idle when an interrupt is acknowledged. When the Handling mode is Level, the Pending state is unchanged when an interrupt is acknowledged.
- **Routing mode** Targeted or 1 of N.
- **Affinity** Specifies the destination of an interrupt.

$D_{LDFPT}$ Each interrupt type has a separate INTID.ID namespace.

$I_{GTDKF}$ For example, PPI 15 is a separate interrupt from LPI 15.

$R_{PSTJC}$ The architecture supports up to 24 bits of INTID.ID namespace for each interrupt type.

$I_{PBBTX}$ An implementation may support fewer than 24 bits of INTID.ID namespace.

$I_{CTQKK}$ The implemented INTID.ID namespace width may vary across interrupt types.

$R_{GYVWB}$ Where relevant, the interrupt type is encoded using a 3-bit field as follows:

- `0b001`: Private Peripheral Interrupts (PPIs)
- `0b010`: Logical Peripheral Interrupts (LPIs)
- `0b011`: Shared Peripheral Interrupts (SPIs)

All other values are reserved.

$R_{TJPHS}$ The INTID is a numerical value constructed from both the interrupt ID and the type as follows:

- Bits[31:29]: Interrupt type
- Bits[28:24]: RES0
- Bits[23:0]: Interrupt ID

$I_{VCLSH}$ For example, an LPI with interrupt ID 1024 can be identified using INTID `0x40000400`.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{DLGBQ}$ The Handling mode of an interrupt specifies whether the Pending state becomes Idle when that interrupt is acknowledged.

When using the GICv5 system architecture, the IRS specifies a separate Trigger mode for SPIs and the IWB specifies a Trigger mode for input wires, and interrupt events generated by an IWB or an IRS may update the Handling mode of the corresponding interrupt to match the Trigger mode.

See Chapter 4 *Interrupt routing service (IRS)* and Chapter 6 *Interrupt Wire Bridge (IWB)* for more information.

$D_{CHFFT}$ LPIs and SPIs are routed to appropriate PEs based on the Routing mode and Affinity. PPIs are private to a PE and therefore always signaled locally on a PE.

$I_{ZPQDY}$ The Routing mode of each LPI and SPI is either Targeted or 1ofN.

$I_{LLQVD}$ Support for 1ofN is optional for the IRI. When the GICv5 system architecture is used, the IRS reports whether 1ofN is supported in IRS_IDR0.ONE_N.

$D_{QHNYP}$ When the Routing mode of an interrupt is 1ofN, the IRI dynamically selects a PE to present the interrupt to, based on information available to the IRI at any point in time.

When the Routing mode of an interrupt is Targeted, the IRI is only permitted to present the interrupt to the PE corresponding to the interrupt Affinity.

See 2.6.2 *LPI and SPI configuration* for more information about configuring the routing of LPIs and SPIs.

### 2.4.1 PE-Private Peripheral Interrupts (PPIs)

$G_{BGLBP}$ GICv5 enables PE local events to be signaled as interrupts without recourse to the IRI.

$D_{PDXVM}$ The GIC CPU Interface Extension defines PPI IDs through 127.

$I_{QTYLP}$ PPIs are generated by the CPU interface, typically by a function within the PE.

$I_{YXRMV}$ For example, the Arm Generic Timers generate a PPI when a timer condition is met.

$R_{LWYBR}$ The mechanism by which PPI sources signal the CPU interface is IMPLEMENTATION DEFINED.

Updates to the PPI Pending state from a PPI source are *autonomous asynchronous events* that complete in finite time. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information.

$I_{DQVMP}$ When a PPI source is programmed using System register writes, even if a direct System register write causes a PPI signal to become asserted or de-asserted, this does not cause an indirect System register write to the ICC_PPI_xPENDR registers. Also in this case, the update from the PPI source to the PPI Pending state is an autonomous asynchronous event that completes in finite time.

$D_{HQXDV}$ Each PPI is private to the PE.

$D_{BWMGN}$ Each PE has a private namespace for physical PPIs. The namespace is common across all Physical Interrupt Domains on the PE.

$I_{KPGFB}$ Because each PPI is private to the PE, and each PE has a private namespace for PPIs, when separate PEs use the same PPI INTID, they specify separate interrupts.

For example, physical PPI 15 on PE 0 is a separate interrupt from physical PPI 15 on PE 1.

$D_{LHMTH}$ If EL2 is implemented, the CPU interface supports virtual PPIs.

$D_{RGQHS}$ Each PE has a separate namespace for virtual PPIs.

$I_{ZTTQT}$ Each virtual PPI is a separate interrupt from the physical PPI with the same INTID on the same PE.

For example, physical PPI 15 on PE 0 is a separate interrupt from virtual PPI 15 on PE 0.

However, a physical PPI can be assigned to the virtual PPI with the same INTID.

See 2.10.1 *Virtual PPIs* for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_ZVRNF The Handling mode of a PPI is not software programmable but is discoverable through a read-only System register.

I_WQBLP In GICv5 the configuration and state of PPIs is held in the CPU Interface and is accessible using the PPI System registers. This is different to earlier versions of the GIC architecture, where the GIC IRI is responsible for PPI state and configuration.

See 9.4 *PPI registers* for more information.

D_JHZKS When a PPI is *implemented*, its state and configuration can be accessed using the PPI System registers.

When a physical PPI is *not implemented*, all of the following are true:

- Access to the corresponding state and configuration using the PPI System registers is RAZ/WI.
- The PPI cannot be made Pending and is not considered for being selected as the candidate HPPI for any Interrupt Domain.

I_FVNFL Whether a PPI is implemented is expressed by the pseudocode function `PPI[] IsImplemented()`. See Chapter C1 *Operational model* for more information.

See also:

- 2.9.1 *Physical PPIs*
- 2.10.1 *Virtual PPIs*

### 2.4.2 Logical Peripheral Interrupts (LPIs)

G_PTPSK LPIs are expected to be provisioned by Hypervisor and OS software according to the needs of the software agent.

Therefore, LPIs have a private namespace for each Interrupt Domain and typically require memory to be allocated for storing their configuration and state in the IRS.

I_ZBQLG LPIs are so called because of the similarity between new LPIs in GICv3 and LPIs in GICv5 are translated by an ITS and often used for interrupt sources that signal their interrupts as MSIs.

The word logical is chosen to reflect the INTID private namespace for each Interrupt Domain.

GICv3 used the words locality-specific for historical reasons.

D_BBVWN Each Physical Interrupt Domain has a private namespace for physical LPIs.

The namespace is common across all PEs for the same Physical Interrupt Domain.

I_PPFTN For example, LPI 15 as seen by PE 0 is the same interrupt as LPI 15 seen by PE 1, in the same Physical Interrupt Domain.

D_BSNNZ The Virtual Interrupt Domain has a separate namespace for virtual LPIs.

The namespace for virtual LPIs is determined by the resident VPE and resident VM.

Each VM has a separate namespace for virtual LPIs. The virtual LPI namespace for a VM is common for all VPEs in a VM.

---

**Note**

This means that when separate VPEs for the same VM are resident across multiple PEs, the Virtual Interrupt Domains across those PEs share the same virtual LPI namespace.

See 2.10.5 *Selecting the resident VPE* for more information.

---

I_FSXHB For example, LPI 15 as seen by VPE 0 is the same interrupt as LPI 15 seen by VPE 1, in the same VM.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 2.4.3 Shared Peripheral Interrupts (SPIs)

GKCZPX SPIs are designed to function without being provisioned by software and without requiring memory to be allocated for storing their configuration and state in the IRI.

SPIs are expected to be configured and provisioned when the system is designed and therefore have a common namespace across the Physical Interrupt Domains which allows firmware to configure the Interrupt Domain that each SPI is assigned to.

SPIs are expected to be used in the following scenarios:

- During early-boot where memory is not available to store interrupt state and configuration.
- When supporting RAS or other error handling interrupts which must work correctly even in the presence of memory system failures.
- In real-time sensitive systems, where the worst-case latency of interrupt delivery and handling must meet pre-determined timing requirements.

DNNDZX The Physical Interrupt Domains have a common namespace for physical SPIs. The namespace is common across all PEs for all Physical Interrupt Domains.

This means that there is a single common namespace for physical SPIs in the system.

DVZHQW The Virtual Interrupt Domain has a separate namespace for virtual SPIs.

The namespace for virtual SPIs is determined by the resident VPE and resident VM.

Each VM has a separate namespace for virtual SPIs. The virtual SPI namespace for a VM is common for all VPEs in a VM.

---

**Note**

This means that when separate VPEs for the same VM are resident across multiple PEs, the Virtual Interrupt Domains across those PEs share the same virtual SPI namespace.

See 2.10.5 *Selecting the resident VPE* for more information.

---

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.5 Inter-Processor Interrupts

$D_{XHRZC}$ The architecture supports inter-processor communications by using *inter-processor interrupts* (IPIs).

IPIs are either SPIs or LPIs configured as Targeted with the Affinity specifying the destination of the IPI.

Arm recommends that interrupts used for IPIs are not signaled by other interrupt sources.

$S_{BDGNX}$ In a Physical Interrupt Domain, an IPI is configured and sent from a source PE to a destination PE as follows:

1. A physical interrupt is selected and configured to target the destination PE.
2. The interrupt Handling mode is configured to Edge.
3. Software running on the source PE issues a `GIC CDPEND` instruction specifying the INTID of the interrupt.

For virtual machines, an IPI is configured and sent from a source VPE to a destination VPE as follows:

1. A virtual interrupt is selected and configured to target the destination VPE.
2. The interrupt Handling mode is configured to Edge.
3. Software running on the source VPE issues a `GIC CDPEND` instruction specifying the INTID of the interrupt.

$I_{SZFVG}$ The effect of making an interrupt Pending completes in finite time.

This means that when an IPI is made pending, if the interrupt meets the conditions to be signaled by the IRI, it is signaled to the target PE or VPE in finite time.

---

**Note**

When an IPI is signaled to its destination, there is no guarantee that it will generate an interrupt exception and be handled by the destination (V)PE. For example, the destination PE could be masking interrupt exceptions or continuously be handling higher priority interrupts.

---

$I_{BXHQK}$ Arm strongly recommends that the system implements a sufficient number of interrupts that are not signaled by other interrupt sources, to service IPIs for each Interrupt Domain.

The GICv5 system architecture allows software in each Interrupt Domain to allocate a sufficient number of LPIs and allows implementations to provide a sufficient number of SPIs for IPIs when appropriate.

$I_{WHWNY}$ Arm expects software to use LPIs allocated by system software to send IPIs.

The architecture also allows the use of SPIs to send IPIs if a sufficient number of SPIs that are not connected to any interrupt sources are available.

$S_{PJWMR}$ Software may allocate the number of required LPIs for each PE and establish a mapping from an LPI INTID.ID to a PE and IPI number.

If software requires *n* IPIs for each PE, and the system supports a maximum of *m* PEs, software would allocate *m* * *n* LPIs for IPIs.

The LPI IDs used for IPIs is managed by software.

However, if software requires a fixed arithmetic mapping for the IPI number on a given PE to an LPI ID, then software can allocate the LPIs 0 through (*m* * *n*) - 1 for IPIs at system boot, and IPI number *x* on PE *y* is then given by (*y* * *n*) + *x*.

If SPIs are used to send IPIs, whether it is possible to create a fixed arithmetic mapping depends on the INTIDs for implemented SPIs that can be used for this purpose.

$S_{FYVVP}$ Software can send IPIs by using the `GIC <domain>AFF, <Xt>`, `GIC <domain>HM, <Xt>` and `GIC <domain>PEND, <Xt>` instructions to configure interrupts and to make them Pending.

Software can configure interrupts for each potential destination PE during boot as follows:

- The interrupt Handling mode is configured to be Edge using the `GIC <domain>HM, <Xt>` instruction.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The interrupt Routing mode is configured to be Targeted and the Affinity can be programmed to the destination PE using the `GIC <domain>AFF, <Xt>` instruction.

At runtime, software can send an IPI to a target PE by issuing the `GIC <domain>PEND, <Xt>` instruction for one of the interrupts targeted to the PE and allocated for IPIs.

See 2.6 *GIC System instructions* for more information about the GIC System instructions.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.6 GIC System instructions

$I_{RSVHM}$ FEAT_GCIE defines the GIC System instructions for the following purposes:

- Configure LPIs and SPIs managed by the IRI.
- Handle pending interrupts in the CPU interface.

$I_{FVJSG}$ The A64 assembly language syntax for GIC System instructions is one of the following:

- `GIC <operation>, <Xt>`.
- `GICR <Xt>, <operation>`.

The `GIC` variant is used when the instruction has the semantics of a write and is an alias of the `SYS` instruction.

The `GICR` variant is used when the instruction has the semantics of a read and is an alias of the `SYSL` instruction.

$I_{ZCVGT}$ The `<operation>` in `GIC <operation>, <Xt>` and `GICR <Xt>, <operation>` has the structure of `<domain><command>`.

Each instruction operates in the Interrupt Domain identified by the `domain` parameter as described below:

| Domain parameter | Interrupt Domain |
|---|---|
| `CD` | Current Interrupt Domain |
| `LD` | Logical Interrupt Domain |
| `VD` | Virtual Interrupt Domain |

Each instruction executes a command specified by a `command` parameter listed below in the identified Interrupt Domain.

| Variant | Command | Name |
|---|---|---|
| GIC | `AFF` | Interrupt set Affinity |
| GIC | `RCFG` | Request interrupt configuration |
| GIC | `DI` | Deactivate interrupt |
| GIC | `DIS` | Interrupt clear Enable |
| GIC | `EN` | Interrupt set Enable |
| GIC | `EOI` | Interrupt Priority drop |
| GIC | `PEND` | Interrupt set/clear Pending state |
| GIC | `PRI` | Interrupt set Priority |
| GICR | `IA` | Interrupt acknowledge |
| GICR | `NMIA` | NMI acknowledge |

- The commands for interrupt configuration are described in 2.6.2 *LPI and SPI configuration*.
- The commands for interrupt handling are described in 2.8 *Interrupt handling*.

`<Xt>` encodes additional parameters specific to the requested operation.

$R_{XCLJC}$ The effects of a GIC System instruction complete in finite time.

$I_{BMZKC}$ GIC System instructions and GSB instructions complete in finite time.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{MFDQT}$ When the GIC System instructions are executed with the `<domain>` parameter as `CD`, the operation applies to the Current Interrupt Domain.

The GIC System instructions are not available when ICH_VCTLR_EL2.V3 is 1 and executing at Exception levels below EL2.

$I_{VNKRX}$ When the GIC System instructions are executed with the `<domain>` parameter as `LD`, the operation applies to the Physical Interrupt Domain selected by the current value of the SCR_EL3.{NSE,NS} fields.

The `LD` parameter is only available at EL3.

$D_{GBYBF}$ An interrupt specified by an INTID is either *reachable* or *unreachable*.

$R_{PCDRJ}$ When a GIC System instruction specifies an unreachable interrupt, it exectutes without generating any interrupt read or write effects.

In this case, all GIC System instructions, except `GIC xDRCFG`, do not rep[illegible] the a[illegible]mpt to access an unreachable INTID.

$R_{JRDTL}$ A physical PPI is reachable when all of the following are true:

- The PPI is implemented.
- The PPI is assigned to the Physical Interrupt Domai[illegible]hat the in[illegible]ruction [illegible]ates in.

A virtual PPI is reachable when the PPI is implemented.

Otherwise, the PPI is unreachable.

$R_{GRSYS}$ A physical SPI is reachable by the CPU inte[illegible]ce when [illegible] of the fo[illegible]owing are true:

- The SPI is implemented by the IRI
- The SPI is assigned to the Physica[illegible]terrupt [illegible]omain that the instruction operates in.

A virtual SPI for a VM is reacha[illegible]e when the [illegible]rtual S[illegible] is provisioned for the resident VM.

Otherwise, the SPI is unreac[illegible]able.

When using the GICv5 [illegible]stem archite[illegible]re, the IRS configuration determines whether an SPI is reachable by the IRS. See 4.8.2 *Phys[illegible]al SPIs* and 4.10.2 *[illegible]rtual SPIs* for more information.

$R_{TTJYH}$ A physical LPI i[illegible]eachable[illegible] by the CPU interface when the LPI is within the configured range for the corresponding Physical Interrupt [illegible]m[illegible]n in the IR[illegible]

A virtua[illegible]LPI [illegible]a VM[illegible]reach[illegible]le when the virtual LPI is provisioned for the resident VM.

Oth[illegible]wise, the [illegible]PI is unreac[illegible]able.

When [illegible]n[illegible]he GICv5 system architecture, the IRS configuration determines whether an LPI is reachable by the IRS. See [illegible]8.1 *Physical LPIs* and 4.10.1 *Virtual LPIs* for more information.

$I_{HDTHV}$ When a GIC S[illegible]tem instruction specifies a virtual SPI or LPI, the VM used to define the namespace for the INTID is determined in one of the following ways:

- If the instruction operation is `VDPEND`, the VM is specified by the register argument passed to the instruction.
- If the instruction domain parameter specifies the Current Interrupt Domain or the Virtual Interrupt Domain, the VM is the resident VM.

When a GIC System instruction operates in the Virtual Interrupt Domain and there is no resident VPE, all virtual LPIs and SPIs are unreachable.

See 2.10.5 *Selecting the resident VPE* for more information about managing the resident VPE.

$I_{HZLFG}$ The following GIC System instructions are related to interrupt handling and access PPI, LPI, or SPI configuration and state:

- `GICR CDIA`.
- `GICR CDNMIA`.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- GIC <domain>DI.

The GIC <domain>DI instruction is treated as a NOP if it specifies an unreachable INTID.

$I_{JNRPR}$ If HCR_EL2.NV is 1, execution at EL1 of a GIC System instruction that specifies VD as the <domain> parameter is trapped to EL2 with ESR_EL2.EC reporting 0x18.

$I_{ZFDJZ}$ A GIC System instruction never successfully executes at EL1 if it specifies the VD as the <domain> parameter. The instruction is either UNDEFINED or trapped to EL2 depending on the value of HCR_EL2.NV.

### 2.6.1 GIC synchronization barriers (GSBs)

$I_{GMHGR}$ The architecture supports the following barrier instructions used to define ordering between effects of GIC System instructions and other effects in the system:

- GSB ACK.
- GSB SYS.

Architectural execution of a GSB instruction guarantees that:

- No load, store, or GIC System instruction appearing in program order after the GSB performs any part of its functionality until the GSB completes, other than:
  - Being fetched from memory and decoded.
  - Reading the general-purpose, SIMD and floating-point, SVE vector or predicate, Special-purpose, or System registers that are directly or indirectly read without causing side-effects.
- For GSB SYS: All Interrupt Effects of all GIC System instructions in program order before the GSB SYS instruction are complete before the completion of the GSB SYS instruction.
- For GSB ACK: All Interrupt Effects of all GICR instructions in program order before the GSB ACK instruction are complete before the completion of the GSB ACK instruction.

See 2.12.2 *GSB instruction semantics* for more information.

$R_{WSSNV}$ When the effects of a GIC System instruction are synchronized by the execution of a GSB instruction, and those effects cause an interrupt to no longer be the HPPI, any resulting de-assertions to the pending state of the IRQ, FIQ, vIRQ, or vFIQ asynchronous exceptions are observed by instructions appearing in program order after the GSB.

---

**Note**

When the effects of a GICR instruction update the pending state of an asynchronous exception, the update is synchronized by the execution of a GSB ACK instruction, regardless of whether the type of the acknowledged interrupt is PPI, LPI, or SPI.

---

See also:

- 2.12 *Interrupt ordering model and synchronization requirements*

### 2.6.2 LPI and SPI configuration

$I_{BFFCW}$ Configuration and state for LPIs and SPIs are managed by the IRI. GIC System instructions are used to update the configuration of an LPIs and SPIs.

$I_{MBNRN}$ When an INTID whose Type is LPI or SPI specifies an ID which is beyond the range reported in ICC_IDR0_EL1.ID_BITS, the unimplemented upper identifier bits are RES0.

$I_{WNZLQ}$ An INTID might be within the implemented INTID width but be unreachable. For example, the system might support 16 bits of INTID but software might have configured a smaller LPI space in the IRI. This is different to specifying an INTID beyond the implemented range, as unimplemented bits of INTID are treated as RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{JGZYC}$ The Affinity of an interrupt is configured by issuing the `GIC <domain>AFF, <Xt>` instruction.

The `IRM` field in `<Xt>` controls the interrupt routing mode. The interrupt routing mode is either Targeted or 1ofN.

When the interrupt is Targeted, the target PE is specified in the `IAFFID` field in `<Xt>`.

If the IRI supports fewer than 16 bits of IAFFID for the PE, unimplemented upper bits are RES0.

$R_{JKYGN}$ When the Affinity of an interrupt is configured by the `GIC <domain>AFF, <Xt>` instruction and the `IRM` field configures the interrupt to be Targeted, all of the following are true:

- For a Physical Interrupt Domain, the `IAFFID` field specifies the PE with the corresponding interrupt Affinity ID.
- For the Virtual Interrupt Domain, the `IAFFID` field specifies the VPE in the VM using the corresponding VPE ID.

$S_{WVYLC}$ Arm expects that Hypervisor software emulates a virtual GICv5 implementation where the emulated ICC_IAFFIDR_EL1.IAFFID value is the VPE ID of the corresponding VPE. This means that to software running in a VM, configuring the Affinity of a virtual interrupt using the virtualized ICC_IAFFIDR_EL1.IAFFID value will have the desired effect.

$I_{WLBBY}$ The `GIC <domain>PEND, <Xt>` instruction has the following effects on the interrupt state:

- If the `Pending` field is cleared to 0, the IRI is requested to clear the Pending state of the interrupt to 0.
- If the `Pending` field is set to 1, the IRI is requested to set Pending state of the interrupt to 1.

$I_{PKRLN}$ The Handling mode of an interrupt can be configured by issuing the `GIC <domain>HM, <Xt>` instruction.

$R_{YCDBY}$ It is IMPLEMENTATION DEFINED whether the Handling mode of an SPI is RO.

If the Handling mode of an SPI is RO, one of the following are true:

- `GIC <domain>HM, <Xt>` instructions targeting the SPI have no effect.
- The corresponding Trigger mode in the IRS is RO.
- The Handling mode corresponds to the Trigger mode.

See 4.8.2 *Physical SPI* for more information.

$I_{TNGVR}$ Interrupt configuration managed by an IRS is queried using the System instruction `GIC <domain>RCFG` and read from the System register ICC_ICSR_EL1.

$R_{NPDVH}$ On executing a `GIC <domain>RCFG` System instruction, the PE requests the configuration and state of the specified INTID from the IRI for the Interrupt Domain identified by the `<domain>` parameter.

If the INTID is reachable, ICC_ICSR_EL1.F is set to 0 and the other fields are populated with the configuration and state of the INTID.

If the INTID is unreachable, ICC_ICSR_EL1.F is set to 1 and the other fields become UNKNOWN.

$I_{GBYBZ}$ Where an instruction results in an update to a System register, as is the case with the `GIC <domain>RCFG` System instruction, explicit synchronization must be performed before the result is guaranteed to be visible. This means that the result of a `GIC <domain>RCFG` System instruction is not guaranteed to be visible in ICC_ICSR_EL1 until after an `ISB` or other context synchronization event.

See also:

- 2.12 *Interrupt ordering model and synchronization requirements*
- 4.6 *Interrupt configuration and state*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.7 Interrupt Prioritization

I_DQGTJ The priority of an interrupt is an unsigned value that is used for the following purposes:

- Selecting which interrupt to signal among multiple interrupts that can be signaled for an Interrupt Domain.
- Priority-based masking of interrupts.
- Configuring an interrupt with a priority value of 0 indicates that the interrupt should be signaled with Superpriority. See 2.9.5 *Physical non-maskable interrupts* for more information about NMIs.
- When there are interrupts that can be signaled for multiple Interrupt Domains, the priority may be used to decide whether an interrupt should be signaled using the IRQ or FIQ interrupt exception. See 2.9.3 *Preemptive interrupts* for more information.

I_ZCBZT The number of implemented priority bits for the physical CPU interface is reported in ICC_IDR0_EL1.PRI_BITS.

Currently, the only permitted number of implemented priority bits is 5, and the priority of an interrupt is always interpreted as a 5 bit unsigned value.

I_VWYWB The IRI may implement fewer priority bits than the CPU interface for an Interrupt Domain. In this case, unimplemented lower order bits are treated as 0 by the IRI. Arm recommends that software only programs priority values within the range supported by the IRI for the Interrupt Domain.

I_CNQYT Some operating systems have requirements for a minimum number of interrupt priority levels. These requirements will be captured as part of the *Arm® Base System Architecture 1.0* [2].

I_YQCPV The priority value of an interrupt is only interpreted within an Interrupt Domain. Priorities of interrupts belonging to separate Interrupt Domains are never compared against one another.

Therefore, the full implemented priority space is available to prioritize interrupts within each Interrupt Domain.

I_GDMTN The priority value of a virtual interrupt is only interpreted to select higher priority interrupts for that VM. The priority value of a virtual interrupt is never compared against the priority value of a physical interrupt. Priorities of interrupts belonging to separate VMs are never compared against one another.

Therefore, the full implemented priority space is available to prioritize interrupts within each VM.

R_XBQTT In the GIC prioritization scheme, lower numbers have higher priority.

For example, an interrupt with priority value 0 is of higher priority than an interrupt with priority value 16.

D_LDCKK The *highest priority pending interrupt* (HPPI) for an Interrupt Domain is the highest priority interrupt selected among the candidate HPPIs.

An HPPI is defined for each Physical Interrupt Domain and the Virtual Interrupt Domain:

- The Group 3 physical HPPI.
- The Secure physical HPPI.
- The Realm physical HPPI.
- The Non-secure physical HPPI.
- The virtual HPPI for the Virtual Interrupt Domain.

D_RKWNS A *candidate HPPI* is a Pending, Inactive, and Enabled interrupt selected among a subset of interrupts for an Interrupt Domain.

The PE considers the following candidate HPPIs for each Interrupt Domain:

- The highest priority Pending, Inactive, and Enabled PPI for the Interrupt Domain.
- The candidate HPPI presented by the IRI for the Interrupt Domain.

R_JXKKZ The state and configuration of a PPI are updated as a result of any of the following:

- A System register write effect.
- An Interrupt Effect generated by a GIC or GICR instruction.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- An autonomous asynchronous event.

When any of these effects complete and result in the PPI meeting the HPPI criteria, the subsequent effects on the HPPI determination complete in finite time.

See 2.9.4 *Physical interrupt signaling* and 2.10.3 *Virtual interrupt signaling* for more information.

$I_{VCCDY}$ The effects of a direct write to a PPI System register without explicit synchronization is not guaranteed to complete in finite time. See $I_{ZLTKB}$ for more information.

$R_{YRGMH}$ When a System register write effect updates the state or configuration of a PPI and this results in the PPI no longer meeting the HPPI criteria, a Context synchronization event guarantees all of the following for instructions appearing in program order after the Context synchronization event:

- The PPI is no longer the HPPI and can therefore not be acknowledged by a `GICR` instruction.
- If the update to the PPI also results in the IRQ, FIQ, vIRQ, or vFIQ asynchronous exceptions no longer being pending, the exception is not taken.

$R_{KXFKH}$ The IRI selects the highest priority Pending, Inactive, and Enabled interrupt for each Interrupt Domain in finite time.

See 4.8.4 *Physical interrupt signaling* and 4.10.4 *Virtual interrupt signaling* for more information about the guarantees provided by the GICv5 system architecture.

See also:

- 2.8 *Interrupt handling*
- 2.8.1 *Interrupt life cycle*
- 2.9 *The physical CPU interface*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 2.8 Interrupt handling

$D_{QWBZR}$ When the CPU interface acknowledges an interrupt, the interrupt's priority at the time of acknowledgement becomes an *active priority*.

$D_{DWSGL}$ Each Interrupt Domain has a *running priority*, which is the highest active priority for that Interrupt Domain.

$I_{FGFPH}$ It is possible to have multiple active priorities at the same time.

For example, consider the following scenario where an interrupt handler is interrupted by an NMI:

1. Interrupt X with priority 16 is acknowledged. Priority 16 becomes an active priority and the running priority of the Interrupt Domain becomes 16.
2. Another interrupt Y with priority 0 is signaled to the PE.
3. Interrupt Y is acknowledged. Priority 0 becomes an active priority and the running priority of the Interrupt domain becomes 0.

At the end of the sequence above, priorities 16 and 0 are active priorities and the running priority for the Interrupt Domain is 0.

$D_{MQKDW}$ When there are no active priorities, the *running priority* is the *Idle priority*. The *Idle priority* is `0xFF`.

$I_{CFYGY}$ Only interrupts with a priority higher than the running priority are signaled by the CPU interface. This prevents a high priority interrupt from being preempted by a low priority interrupt.

$D_{VHCMF}$ The CPU interface performs a *Priority drop* when software issues a `GIC CDEOI` instruction, and the running priority is not the Idle priority.

$R_{KWMNP}$ On a Priority drop, the highest active priority for the Interrupt Domain stops being active.

$I_{FGKMJ}$ After a Priority drop, because the running priority is defined as the highest active priority, the running priority becomes one of the following:

1. The highest active priority for which there has not been a Priority drop.
2. The Idle priority.

$I_{XZFHX}$ This allows pending interrupts with the previous running priority to be signaled by the CPU interface.

$I_{QFZWR}$ Active priorities drop being active in priority order, from the highest priority to the lowest priority.

$I_{JFQSN}$ GICv5 removes support for the binary point registers used in GICv3 to split the priority field into a preemption level and a group priority.

$I_{FYZVD}$ Software handles an interrupt signaled by the CPU interface as follows:

1. Software acknowledges the HPPI for the Current Interrupt Domain and obtains its INTID.
2. Software uses the INTID to locate the handler function for the acknowledged interrupt.
3. Software performs a Priority drop when it is ready to receive other interrupts of the same priority as the acknowledged interrupt.
4. Software deactivates the acknowledged interrupt once it is ready to receive that interrupt again.

$I_{DFWSM}$ FEAT_GCIE defines GIC System instructions to handle interrupts in the CPU interface. Each instruction executes a command that applies to all interrupt types in the Current Interrupt Domain. The commands are listed below:

| Instruction | Command | Name |
|---|---|---|
| `GICR` | `IA` | Interrupt acknowledge |
| `GICR` | `NMIA` | NMI acknowledge |
| `GIC` | `EOI` | Interrupt priority drop |
| `GIC` | `DI` | Interrupt deactivate |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The System instructions for interrupt handling are listed below and described in Chapter 8 *System instructions*.

$I_{ZYDVS}$ The architecture does not define the order in which the Priority Drop and Interrupt deactivate are performed by software.

$I_{WDLPR}$ The `GICR CDIA` instruction acknowledges the HPPI in the Current Interrupt Domain if there is an HPPI with Sufficient priority and it is not an NMI.

The result of the instruction is returned in the `<Xt>` register.

The `VALID` field in `<Xt>` register indicates whether an interrupt was acknowledged by the `GICR CDIA` instruction.

When the `VALID` field is 1, the HPPI was acknowledged.

When the `VALID` field is 0, the HPPI was not acknowledged.

$I_{XBBNJ}$ The `GICR CDNMIA` system instruction acknowledges the HPPI in the Current Interrupt Domain if there is an HPPI with Sufficient priority and it is an NMI.

The result of the instruction is returned in the `<Xt>` register.

The `VALID` field in `<Xt>` register indicates whether the HPPI was acknowledged.

When the `VALID` field is 1, the HPPI was acknowledged.

When the `VALID` field is 0, the HPPI was not acknowledged.

$R_{WNKKW}$ When an interrupt is acknowledged, all of the following are true:

- The interrupt becomes Active.
- If the interrupt Handling mode is Edge, its Pending state is cleared.
- The interrupt's priority becomes active and the running priority for the Current Interrupt Domain is updated.
- The INTID of the interrupt is returned in the INTID and ID fields of the `<Xt>` register.

$I_{HBGJV}$ The `GIC CDEOI` System instruction performs a priority drop of the running priority in the Current Interrupt Domain.

$I_{RYBDB}$ Unlike earlier versions of the GIC architecture, the `GIC CDEOI` System instruction does not take an INTID as an argument.

$I_{WPSSR}$ The `GIC CDDI` System instruction performs an Interrupt deactivate for the specified INTID in the Current Interrupt Domain.

### 2.8.1 Interrupt lifecycle

$R_{TCZMV}$ Figure 2.2 shows the interrupt state machine for interrupts whose Handling mode is Edge.

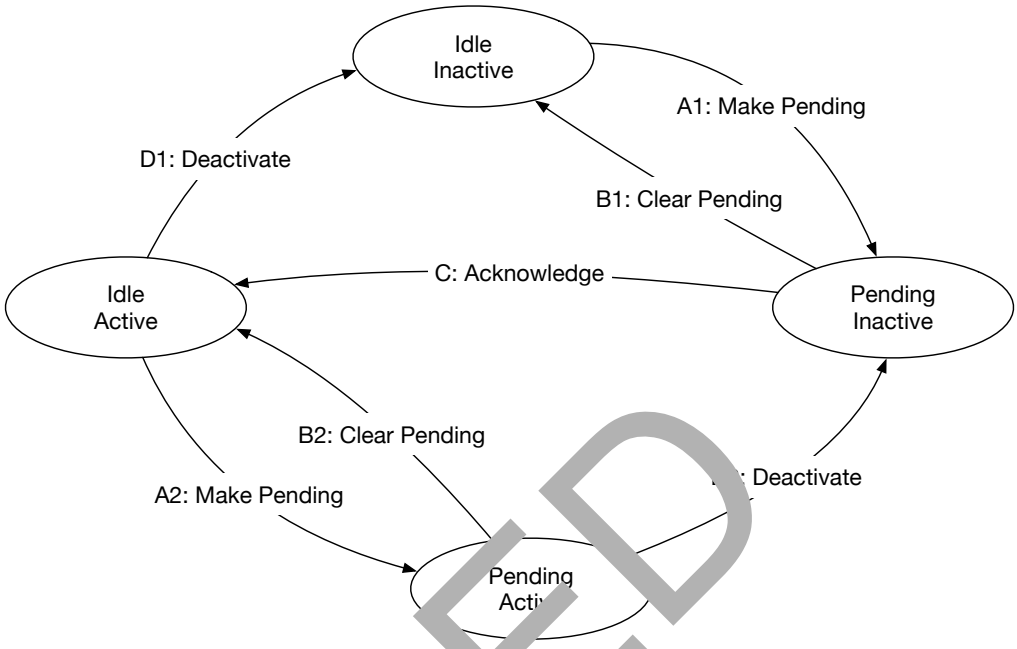
Figure 2.3 shows the interrupt state machine for interrupts whose Handling mode is Level.

The interrupt Handling mode determines the effects on the interrupt state when the CPU Interface acknowledges the interrupt as follows:

- For Edge interrupts, all of the following are true:
  - The Pending state is atomically cleared to Idle.
  - The Active state is set to Active.
- For Level interrupts, The Active state is set to Active.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential




**e 2.2. nterrup tate machine for Edge interrupts.**

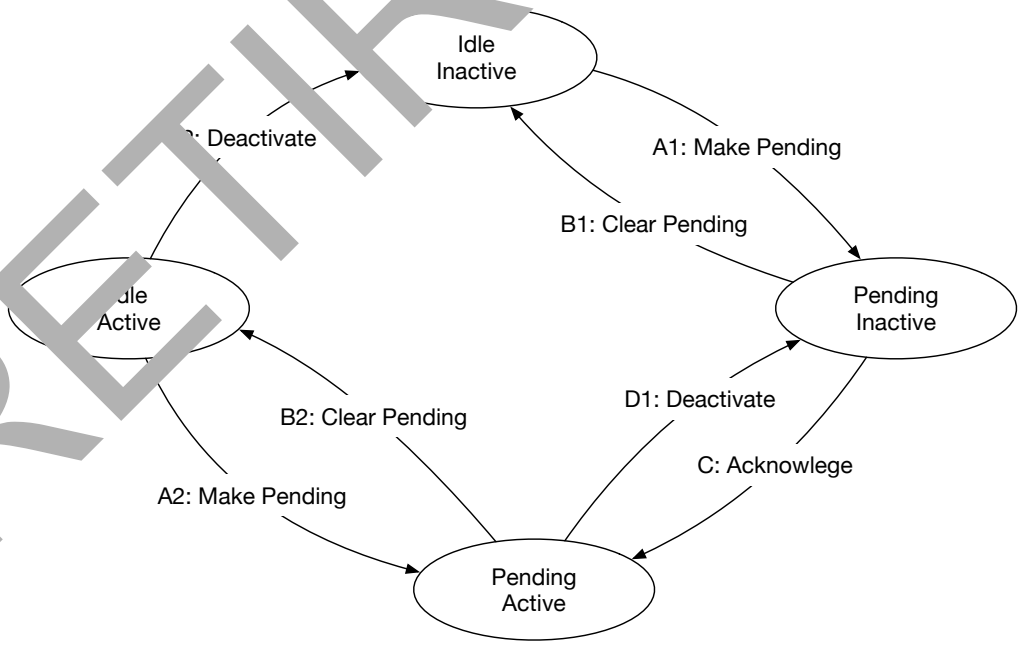



**Figure 2.3: Interrupt state machine for Level interrupts.**

The transitions shown in the figures can be caused by the following reasons:

- **Transition A1 or A2: Make Pending**

  This transition occurs in one of the following situations:

  – An LPI or SPI is signaled in the IRI or requested to be made Pending by software.

- **Transition B1 or B2: Clear Pending**

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

This transition occurs in one of the following situations:

– A level-sensitive LPI or SPI signal is de-asserted in the IRI or the Pending state is requested to be cleared by software.

- **Transition C: Acknowledge**

  This transition occurs when the CPU Interface acknowledges the Interrupt.

  For Edge interrupts, the Pending state is cleared, also known as *consumed*, as part of acknowledging the interrupt.

  For Level interrupts, the Pending state is not affected as part of acknowledging the interrupt.

- **Transition D1 or D2: Deactivate**

  This transition occurs when the CPU Interface deactivates the Interru

$I_{DMXLN}$ An interrupt cannot be set to Active using any other GIC instructio than `GICR IA` and `GICR CDNMIA`. An interrupt can be deactivated from any PE. Deactivating an interrupt w h is already I ctive has no effect.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# 2.9 The physical CPU interface

## 2.9.1 Physical PPIs

$R_{RVCZW}$ A physical PPI is only considered for being selected as the candidate HPPI in the Physical Interrupt Domain it is assigned to.

$R_{BHPGS}$ For each PPI, it is IMPLEMENTATION DEFINED whether it is implemented. See $D_{JHZKS}$ for more information.

$I_{GFBQR}$ It is permitted for a PPI to be implemented without being connected to a PPI source. When an implemented PPI is not connected to a PPI source, it can only be made Pending as the result of a write to ICC_PPI_SPENDR<n>_EL1.

$D_{HVJHM}$ PPIs in the range of IDs 0 through 63 are referred to as *architected PPIs*.

$I_{SHHCL}$ The assignment of an architected PPI to a PPI source is defined by the architecture. Unassigned architected PPIs are Reserved.

$R_{FSFSL}$ The assignment of PPIs in the range of IDs 64 through 127 to their PPI sources is IMPLEMENTATION DEFINED.

$I_{YGJWQ}$ The assignment of a physical PPI to a Physical Interrupt Domain is done as follows:

- Architected PPIs are assigned by programming ICC_PPI_DOMAINR0_EL3 and ICC_PPI_DOMAINR1_EL3.
- IMPLEMENTATION DEFINED PPIs are assigned by programming ICC_PPI_DOMAINR2_EL3 and ICC_PPI_DOMAINR3_EL3.

$I_{GRYHX}$ Access to the state and configuration of a physical PPI using an ICC_PPI_ System register is only permitted when the PPI is assigned to the Current Interrupt Domain as defined. Otherwise, the access is RAZ/WI for that PPI.

Attempts to deactivate the PPI by executing a GIC CDDI System instruction when the PPI is not assigned to the corresponding Interrupt Domain are treated as NOPs.

$R_{RHCDS}$ When EL3 is not implemented, each physical PPI is assigned to the Interrupt Domain identified by the Effective value of SCR_EL3.NS.

$R_{XDVCM}$ The assignments of architected PPIs to their PPI sources are listed in Table 2.7.

**Table 2.7: Architected PPI ID assignment**

| ID | Name | Source |
|---|---|---|
| 31 | TRBIRQ | Trace Buffer Unit |
| 30 | CNTP | EL1 Physical Timer |
| 29 | CNTPS | EL3 Physical Timer |
| 28 | CNTHV | Non-secure EL2 Virtual Timer |
| 27 | CNTV | EL1 Virtual Timer |
| 26 | CNTHP | Non-secure EL2 Physical Timer |
| 25 | GICMNT | GIC maintenance interrupt |
| 24 | CTIIRQ | Generic CTI interrupt trigger event |
| 23 | PMUIRQ | PMU overflow interrupt request |
| 22 | COMMIRQ | Debug communication channel |
| 21 | PMBIRQ | Profiling Buffer management interrupt request |
| 20 | CNTHPS | Secure EL2 Physical Timer |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| ID | Name | Source |
|---|---|---|
| 19 | CNTHVS | Secure EL2 Virtual Timer |
| 15 | HACDBSIRQ | Hardware accelerator for cleaning Dirty state interrupt |
| 3 | SW_PPI | Reserved for software usage. |
| 2 | NS_DB_PPI | Doorbell PPI for a Non-secure Non-preemptive interrupt |
| 1 | RL_DB_PPI | Doorbell PPI for a Realm Non-preemptive interrupt |
| 0 | S_DB_PPI | Doorbell PPI for a Secure Non-preemptive interrupt |

RCFSKX The Handling modes of architected PPIs are listed in Table 2.8.

**Table 2.[illegible] Architected PPI ID Handling modes**

| ID | Name | Handling Mode |
|---|---|---|
| 31 | TRBIRQ | Level |
| 30 | CNTP | Level |
| 29 | CNTPS | Level |
| 28 | CNTHV | Level |
| 27 | CNTV | Level |
| 26 | CNTHP | Level |
| 25 | GICMNT | Level |
| 24 | CTIIRQ | IMPLEMENTATION DEFINED |
| 23 | PMUIRQ | Level |
| 22 | COMMIRQ | Level |
| 21 | PBIRQ | Level |
| 20 | CNTHPS | Level |
| 19 | CNTHVS | Level |
| 15 | HACDBSIRQ | Level |
| 3 | SW_PPI | Edge |
| 2 | NS_DB_PPI | Level |
| 1 | RL_DB_PPI | Level |
| 0 | S_DB_PPI | Level |

IRRGLL The configuration and state of PPIs is held in the PE System registers.

For physical PPIs, the following registers are used:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Table 2.9: Physical PPI System registers**

| Register | Name |
|---|---|
| ICC_PPI_SACTIVER<n>_EL1 | Set interrupt Active |
| ICC_PPI_CACTIVER<n>_EL1 | Clear interrupt Active |
| ICC_PPI_SPENDR<n>_EL1 | Set interrupt Pending |
| ICC_PPI_CPENDR<n>_EL1 | Clear interrupt Pending |
| ICC_PPI_ENABLER<n>_EL1 | Interrupt Enable and Disable |
| ICC_PPI_PRIORITYR<n>_EL1 | Interrupt Priority |
| ICC_PPI_DOMAINR<n>_EL3 | Physical Interrupt Domain allocation |
| ICC_PPI_HMR<n>_EL1 | Interrupt Handling mode (Level or Edge) |

$I_{RBPKR}$ The Active state of a PPI can also be updated as a side effect of executing one of the following GIC System instructions:

- `GICR CDIA`.
- `GICR CDNMIA`.
- `GIC CDDI`.
- `GIC VDDI`.
- `GIC LDDI`.

These System instructions are used during interrupt handling and are common across interrupt types.

$I_{ZLTKB}$ A Context Synchronization event is required before software can rely on the effects of direct writes to a PPI System register to affect instructions appearing in program order after the direct write. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information. This applies to the PPI System registers listed in the following tables:

- Table 2.9
- Table 2.15
- Table ...

$S_{XGNGQ}$ A PPI might be shared between multiple Security states, and in this case, EL3 is expected to save and restore the configuration of the PPI during a change of Security state.

$I_{CPDGZ}$ A physical PPI is permitted to be a candidate HPPI for a Physical Interrupt Domain if and only if all of the following are true:

- ICC_PPI_ENABLER<n>_EL1.EN is 1.
- ICC_PPI_SPENDR<n>_EL1.PEND is 1.
- ICC_PPI_SACTIVER<n>_EL1.ACTIVE is 0.

$I_{HTMYK}$ The CPU interface selects the candidate HPPI among the PPIs for each Physical Interrupt Domain by selecting the highest priority PPI that meets the physical candidate HPPI selection criteria.

$R_{JSQBW}$ If more than one PPI meets the physical candidate HPPI selection criteria for a Physical Interrupt Domain, it is IMPLEMENTATION DEFINED which of those PPIs becomes the candidate HPPI.

$I_{QMBYG}$ Any PPI source may generate an IMPLEMENTATION DEFINED wake event for the PE that they are connected to. The architecture does not specify when a PPI source may generate a wake event.

$I_{HYVBT}$ Examples of when a PPI source may generate an IMPLEMENTATION DEFINED wake event include the following:

- A generic timer is kept powered on while the PE is in low-power state and generates a wake event when the timer condition is asserted.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The Cross-Trigger interface is kept powered on while the PE is in a low-power state and asserts an output trigger event.

### 2.9.2 Physical priority masking

$D_{MLTJM}$ The *Physical Priority Mask* is defined as follows:

- The Physical Priority Mask for a Non-EL3 Interrupt Domain is the value stored in the banked copy of ICC_PCR_EL1.PRIORITY for that Interrupt Domain.
- The Physical Priority Mask for the EL3 Interrupt Domain is the value stored in ICC_PCR_EL3.PRIORITY.

$I_{RGRBD}$ See 9.1 *Synchronization requirements for GICv5 System registers* for more information about synchronization requirements for updates to the Physical Priority Mask.

$D_{XMBQZ}$ The *physical running priority* is defined as follows:

- The physical running priority for a Non-EL3 Interrupt Domain is the highest active priority stored in the banked copy of ICC_APR_EL1 for that Interrupt Domain.
- The physical running priority for the EL3 Interrupt Domain is the highest active priority stored in ICC_APR_EL3.

$I_{FWVXZ}$ The physical running priority for the Current Physical Interrupt Domain is reported in ICC_HAPR_EL1.PRIORITY.

$D_{MSQKF}$ A physical interrupt has *Sufficient priority* to be signaled when one of the following are true:

- The priority of the interrupt is higher than the physical running priority for the Physical Interrupt Domain.
- The priority of the interrupt is equal to or higher than the Physical Priority Mask for the Physical Interrupt Domain.

$I_{QYFVP}$ The physical running priority is defined for each Physical Interrupt Domain even though ICC_HAPR_EL1 only shows the running priority for the Current Physical Interrupt Domain.

The physical running priority is a factor in determining whether an interrupt has Sufficient priority in an Interrupt Domain for each Physical Interrupt Domain, regardless of whether an Interrupt Domain is the Current Physical Interrupt Domain.

For example, when the Current Physical Interrupt Domain is the Non-secure Interrupt Domain, an interrupt in the Secure Interrupt Domain will only have Sufficient priority if the interrupt's Priority is higher than the highest active priority in the Secure banked copy of ICC_APR_EL1.

$I_{WLVKM}$ ICC_DOMAINHPPIR_EL1 reports if there is an HPPI of Sufficient priority for each Non-EL3 Interrupt Domain.

### 2.9.3 Preemptive interrupts

$G_{LTFJK}$ When the Current Physical Interrupt Domain is not the EL3 Interrupt Domain, and there is an HPPI of Sufficient priority for another Physical Interrupt Domain, the PE may be configured to take one of the following actions:

- The Current Physical Interrupt Domain is preempted in order to switch to the Physical Interrupt Domain of the HPPI so that it can be handled immediately. See also 2.9.4 *Physical interrupt signaling*.
- The Current Physical Interrupt Domain is notified through a separate interrupt that there is an HPPI for a different Physical Interrupt Domain, and the Currrent Physical Interrupt Domain can switch to the other Interrupt Domain when appropriate as defined by a software policy. See also 2.9.6 *Doorbell PPIs*.

$D_{MQRSB}$ The Interrupt Domain selected by ICC_CR0_EL3.PID is called the *Preemptive Interrupt Domain*.

$I_{PJKYP}$ When the value of ICC_CR0_EL3.PID corresponds to an Interrupt Domain that is not implemented, the PE behaves as if there is no Preemptive Interrupt Domain.

$I_{VFWFW}$ ICC_CR0_EL1.PID indicates whether the Physical Interrupt Domain associated with the Security state selected by SCR_EL3.{NSE,NS} is the Preemptive Interrupt Domain.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

DNNBML The interrupt priority range of the Preemptive Interrupt Domain is split into two. The priority value used for the split is called the *Interrupt Preemptive Priority Threshold (IPPT)*. ICC_CR0_EL1.IPPT defines the Interrupt Preemptive Priority Threshold for each Physical Interrupt Domain.

IRLTDB The value of ICC_CR0_EL1.IPPT has no effect on the signaling of interrupts for an Interrupt Domain which is not the Preemptive Interrupt Domain.

DKXXGB A physical interrupt is a *Preemptive interrupt* when one of the following is true:

- The interrupt belongs to the EL3 Interrupt Domain and the Current Physical Interrupt Domain is not the EL3 Interrupt Domain.
- All of the following are true:
  - The Current Physical Interrupt Domain is not the Preemptive Interrupt Domain.
  - The interrupt belongs to the Preemptive Interrupt Domain.
  - The priority of the interrupt is higher than the IPPT of the Preemptive Interrupt Domain.

Otherwise, the interrupt is a *Non-preemptive interrupt*.

SYFVLN For example, on a system with Secure and Non-secure Security states, the Secure Interrupt Domain can configure a subset of its interrupts as Preemptive interrupts and the remaining subset as Non-preemptive interrupts:

- The Preemptive interrupts preempt execution in the Non-secure Interrupt Domain before being handled in the Secure Interrupt Domain.
- The Non-preemptive interrupts allow the Non-secure Interrupt Domain to yield execution control to the Secure Interrupt Domain where they are handled.

### 2.9.4 Physical interrupt signaling

IWVKDZ The CPU interface determines the physical HPPI for each Physical Interrupt Domain and performs the following actions:

1. Applies priority masking based on software configured priority masks and the running priority.
2. Determines whether the physical HPPIs signals the IRQ or FIQ interrupt to the PE.
3. Determines whether the interrupt is signaled with Superpriority.

This process is illustrated in Figure 2.4:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

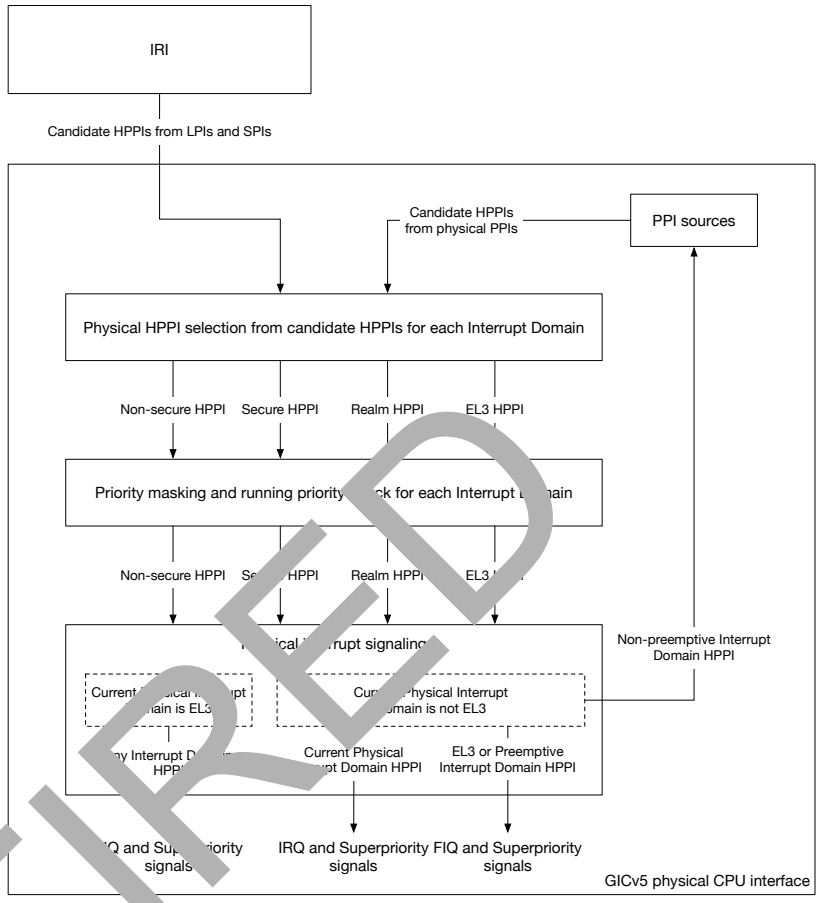



**Figure 2.4: Physical interrupt determination**

$R_{ZSWWN}$ The CPU interface determines whether there is an HPPI for a Physical Interrupt Domain when the value of ICC_CR0_ELx.EN is 1 for the Interrupt Domain.

The HPPI for an Interrupt Domain is determined by selecting the interrupt with the highest priority from the following:

- The candidate HPPI presented by the IRI for the Interrupt Domain.
- The candidate HPPI selected among the physical PPIs for the Interrupt Domain.

If ICC_CR0_ELx.EN is 0, the CPU interface behaves as if there are no candidate HPPIs for the Interrupt Domain.

There is no HPPI for an Interrupt Domain if any of the following are true:

- ICC_CR0_ELx.EN is 0 for the Interrupt Domain.
- There are no candidate HPPIs for the Interrupt Domain.

$R_{VVBPS}$ If the candidate HPPI presented by the IRI has the same priority as the candidate HPPI selected among the PPIs for a Physical Interrupt Domain, it is IMPLEMENTATION DEFINED which of the two candidate HPPIs is selected as the HPPI for the Physical Interrupt Domain.

$R_{XNDSJ}$ When there is at least one candidate HPPI for a Physical Interrupt Domain, the CPU interface determines the HPPI in finite time.

When there is a change to the candidate HPPIs due to updates to the configuration and state of the physical PPIs, or due to the IRI presenting a new candidate HPPI for the Interrupt Domain, the CPU interface redetermines the HPPI for the Physical Interrupt Domain in finite time.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{QLGBG}$ The CPU interface signals the FIQ interrupt when one of the following is true:

- There is a Preemptive interrupt of Sufficient priority.
- All of the following are true:
  - The Current Physical Interrupt Domain is the EL3 Interrupt Domain.
  - There is an HPPI of Sufficient priority for any Interrupt Domain.

$R_{ZGHMN}$ The CPU interface signals the IRQ interrupt when all of the following are true:

- The Current Physical Interrupt Domain is a Non-EL3 Interrupt Domain.
- There is no Preemptive interrupt of Sufficient priority.
- There is an HPPI of Sufficient priority for the Current Physical Interrupt Domain.

$I_{SXXCJ}$ The CPU interface never signals the IRQ and FIQ interrupts at the same time.

$I_{RYDXC}$ The following table illustrates the physical interrupt signal asserted by the CPU interface when the PE is executing at any Exception level below EL3:

| EL3 HPPI of Sufficient priority exists | Non-Current Preemptive Domain HPPI of Sufficient priority exists | Current Domain HPPI of Sufficient priority exists | Interrupt signal |
|---|---|---|---|
| Yes | x | x | FIQ |
| No | Yes | x | FIQ |
| No | No | Yes | IRQ |
| No | No | No | n/a |

x: The value does not have any impact on which interrupt is signaled.

$I_{LTYGD}$ The following table illustrates the type of physical interrupt signaled by the CPU interface when the PE is executing at EL3:

| EL3 HPPI of Sufficient priority exists | Non-EL3 Preemptive HPPI of Sufficient priority exists | Any non-Preemptive HPPI of Sufficient priority exists | Interrupt signal |
|---|---|---|---|
| Yes | x | x | FIQ |
| No | Yes | x | FIQ |
| No | No | Yes | FIQ |
| No | No | No | n/a |

x: The value does not have any impact on which interrupt is signaled.

$S_{MKKWV}$ Software executing at EL3 can read ICC_DOMHPPIR_EL3 to determine if there are HPPIs of Sufficient priority for other Interrupt Domains. For example, this can be useful after taking the FIQ exception to determine what may have caused the exception when execution of the `GICR CDIA` System instruction returns VALID as 0 in the `<Xt>` register.

This may also be useful in certain power management scenarios to determine the idle state of the system.

$I_{HDXLH}$ Whether there is an HPPI of Sufficient priority is reported by the following registers based on the current Exception level:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- When the current Exception level is EL2 or EL1, ICC_HPPIR_EL1 reports whether there is an HPPI of Sufficient priority for the Current Physical Interrupt Domain.
- When the current Exception level is EL3, ICC_HPPIR_EL1 reports whether there is an HPPI of Sufficient priority for the Logical Interrupt Domain as selected by SCR_EL3.{NSE,NS}.

If there is an HPPI of Sufficient priority, it also reports the INTID of that HPPI.

$S_{DWXKH}$ EL3 firmware can use ICC_DOMHPPIR_EL3 to determine if a Security state has an HPPI of Sufficient priority. By switching SCR_EL3.{NSE,NS} to that Security state and subsequently reading ICC_HPPIR_EL1, the firmware can identify the INTID of that HPPI. EL3 firmware can use this information to switch to the software component responsible for handling the HPPI in the Security state.

$I_{GVRTZ}$ In GICv3, the ICx_HPPIRx_EL1 registers indicate the highest priority pending interrupt for the corresponding interrupt group irrespective of whether the interrupt has sufficient priority to be signaled to the PE.

GICv5 changes this behavior such that the ICx_HPPIR_EL1 and IC_DOMHPPIR_EL3 registers return information for an HPPI only if it has Sufficient priority to be signaled.

Arm is not aware of any software component that relies on the GICv3 behavior. The GICv5 behavior helps simplify the PE interface architecture as both the logic for acknowledging an HPPI and reporting information about an HPPI rely on the same condition of Sufficient priority determination.

$I_{SYCYB}$ ICC_HPPIR_EL3 reports whether there is an HPPI of Sufficient priority for the EL3 Interrupt Domain.

$I_{SXGQC}$ In a system which only supports a single Interrupt Domain, the CPU interface never signals the FIQ interrupt.

### 2.9.5 Physical non-maskable interrupts

$R_{CSBDX}$ When there is an HPPI of Sufficient priority for the Current Physical Interrupt Domain and the Current Physical Interrupt Domain is not the EL3 Interrupt Domain, the IRQ interrupt is signaled to the PE with *Superpriority*[1] when the priority of the HPPI is 0.

When there is an HPPI of Sufficient priority for the Current Physical Interrupt Domain and the Current Physical Interrupt Domain is the EL3 Interrupt Domain, the FIQ interrupt is signaled to the PE with Superpriority when the priority of the HPPI is 0.

When there is a Preemptive interrupt of Sufficient priority for the EL3 Interrupt Domain, it is IMPLEMENTATION DEFINED whether the FIQ interrupt is signaled to the PE with Superpriority when the priority of the Preemptive interrupt is 0.

Otherwise, the CPU interface does not signal the interrupt to the PE with Superpriority.

---

**Note**

An interrupt for the EL3 Interrupt Domain is only a Preemptive interrupt when the Current Physical Interrupt Domain is not the EL3 Interrupt Domain. This means that in the rule above, it is only IMPLEMENTATION DEFINED whether the FIQ interrupt is signaled to the PE with Superpriority when executing below EL3.

---

$D_{SDQJW}$ The physical HPPI is an *NMI* when all of the following are true:

- The interrupt is signaled with Superpriority.
- SCTLR_ELx.NMI is 1 for the target Exception level ELx.

$I_{RPSFK}$ If the priority of the HPPI is 0 and SCTLR_ELx.NMI is 0 for the target Exception level ELx, the HPPI is not an NMI.

$I_{DLGVX}$ An NMI can never be acknowledged by executing the `GICR CDIA` instruction.

Similarly, an interrupt that is not an NMI can never be acknowledged by executing the `GICR CDNMIA` instruction. This means that when SCTLR_ELx.NMI is 0 for the current Exception level, the `GICR CDNMIA` instruction cannot acknowledge an interrupt.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{SPWCF}$ When FEAT_GCIE is implemented, following a Context synchronization event as the result of taking or returning from an exception, changes to whether the CPU interface has asserted or de-asserted the IRQ and FIQ signal for the Interrupt Domain have taken effect and can be observed by a read of ISR_EL1.

See also:

- Chapter C1 *Operational model*

### 2.9.6 Doorbell PPIs

$R_{QCVFJ}$ Doorbell PPIs are level-sensitive PPIs.

$R_{PNYKZ}$ There is a separate Doorbell PPI for every Non-EL3 Interrupt Domain as described in Table 2.12.

**Table 2.12: Doorbell PPI definitions**

| Non-EL3 Interrupt Domain | Doorbell PPI name |
|---|---|
| Secure | S_DB_PPI |
| Realm | RL_DB_PPI |
| Non-secure | NS_DB_PPI |

$R_{GSHXH}$ The level of the NS_DB_PPI is asserted when all of the following are true:

- The Current Physical Interrupt Domain is one of the following:
  - Secure Interrupt Domain.
  - Realm Interrupt Domain.
- The NS_DB_PPI is assigned to the Current Physical Interrupt Domain.
- There is a Non-preemptive physical HPPI of Sufficient priority for the Non-secure Interrupt Domain.

Otherwise, the level of the NS_DB_PPI is not asserted.

The level of the S_DB_PPI is asserted when all of the following are true;

- The Current Physical Interrupt Domain is one of the following:
  - Non-secure Interrupt Domain.
  - Realm Interrupt Domain.
- The S_DB_PPI is assigned to the Current Physical Interrupt Domain.
- There is a Non-preemptive physical HPPI of Sufficient priority for the Secure Interrupt Domain.

Otherwise, the level of the S_DB_PPI is not asserted.

The level of the RL_DB_PPI is asserted when all of the following are true;

- The Current Physical Interrupt Domain is one of the following:
  - Non-secure Interrupt Domain.
  - Secure Interrupt Domain.
- The RL_DB_PPI is assigned to the Current Physical Interrupt Domain.
- There is a Non-preemptive physical HPPI of Sufficient priority for the Realm Interrupt Domain.

Otherwise, the level of the RL_DB_PPI is not asserted.

$R_{LWLPK}$ When the level of a Doorbell PPI is asserted, it is asserted in finite time.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

RYSCLK Table 2.13 lists the conditions for when a Doorbell PPI is implemented.

**Table 2.13: Doorbell PPI implementation**

| Doorbell PPI name | Condition |
|---|---|
| S_DB_PPI | EL3 and Secure state are implemented |
| RL_DB_PPI | FEAT_RME is implemented |
| NS_DB_PPI | EL3 is implemented |

IWGXJR When the Current Physical Interrupt Domain is a Non-EL3 Interrupt Domain, Table 2.14 describes when the level of a Doorbell PPI is asserted.

**Table 2.14: Doorbell PPI usage**

| Current Physical Interrupt Domain | Doorbell PPI | Asserted when |
|---|---|---|
| Non-secure | S_DB_PPI | There is a Secure Non-preemptive HPPI of Sufficient priority. |
| Non-secure | RL_DB_PPI | There is a Realm Non-preemptive HPPI of Sufficient priority. |
| Secure | NS_DB_PPI | There is a Non-secure Non-preemptive HPPI of Sufficient priority. |
| Secure | RL_DB_PPI | There is a Realm Non-preemptive HPPI of Sufficient priority. |
| Realm | S_DB_PPI | There is a Secure Non-preemptive HPPI of Sufficient priority. |
| Realm | NS_DB_PPI | There is a Non-secure Non-preemptive HPPI of Sufficient priority. |

Multiple Doorbell PPIs can be asserted at the same time. For example, if the Current Physical Interrupt Domain is the Non-secure Interrupt Domain, and there is a Non-preemptive HPPI of Sufficient priority for both the Secure and Realm Interrupt Domains, then the levels of both S_DB_PPI and RL_DB_PPI are asserted.

SFKVPW When the Current Physical Interrupt Domain is a Non-EL3 Interrupt Domain, if a Doorbell PPI for another Non-EL3 Interrupt Domain is implemented, it must be assigned to the Current Physical Interrupt Domain so that it can be signaled and handled in the Current Physical Interrupt Domain. This is done by software executing at EL3 as follows:

- Prior to a switch to the Non-secure Interrupt Domain from the EL3 Interrupt Domain, the S_DB_PPI and RL_DB_PPI are assigned to the Non-secure Interrupt Domain.
- Prior to a switch to the Secure Interrupt Domain from the EL3 Interrupt Domain, the NS_DB_PPI and RL_DB_PPI are assigned to the Secure Interrupt Domain.
- Prior to a switch to the Realm Interrupt Domain from the EL3 Interrupt Domain, the S_DB_PPI and NS_DB_PPI are assigned to the Realm Interrupt Domain.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$S_{QFPBH}$ Software can manage Doorbell PPIs the same way as managing any other PPIs, including individually enabling and disabling them, and configuring their priority. Doorbell PPIs can be used to yield control to another Security State at an opportune time according to the scheduling policy in the current Security state.

$R_{TCBST}$ On taking an exception from Exception levels below EL3 to EL3, the CPU interface completes the following sequence:

1. The levels of all Doorbell PPIs are de-asserted.
2. The HPPI for each Physical Interrupt Domain is re-evaluated.

See also:

- 2.9.3 *Preemptive interrupts*.
- 2.9.4 *Physical interrupt signaling*.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.10 The virtual CPU interface

I_XVDWC When HCR_EL2.IMO is 1 and ICH_VCTLR_EL2.V3 is 0, the GICv5 virtual CPU interface is used and all of the following are true:

- When executing at EL1, all of the following are true:
  - GICv5 System instruction that specify the Current Interrupt Domain as the domain parameter operate in the Virtual Interrupt Domain.
  - Accesses to GICv5 System registers that share an encoding between the ICC_* and ICV_* registers access the ICV_* registers.
  - Accesses to GICv3.3 Group 0, Group 1, and Common System registers are UNDEFINED.
- Accesses to ICH_VMCR_EL2 use the GICv5 version of the register.

See Chapter 8 *System instructions* and Chapter 9 *System registers* for more information.

S_KFGFJ The Hypervisor at EL2 configures HCR_EL2.IMO to control whether the virtual or physical CPU interface is accessed at EL1.

I_FCTBZ The value of HCR_EL2.FMO has no effect on the operation of the GICv5 CPU interface when Legacy operation is disabled because of the following reasons:

- Since SCR_EL3.FIQ is RES1 if EL3 is implemented, physical FIQ interrupts are taken to EL3.
- When Legacy operation is disabled, the GICv5 virtual CPU interface does not signal the virtual FIQ interrupt.

### 2.10.1 Virtual PPIs

G_WWRGJ The architecture supports signaling virtual PPIs to software running at Exception levels below EL2. Virtual PPIs may be signaled by hypervisor software emulating a PPI source or directly injected using the corresponding physical PPI.

R_GZRNG For each implemented physical PPI, if EL2 is implemented, the corresponding virtual PPI with the same ID is implemented.

I_QFGQN For each implemented PPI, the physical and virtual PPI with the same INTID are separate interrupts.

I_RQBBW If the Virtual CPU interface is supported, virtual PPIs are configured using the following registers:

**Table 2.15: Virtual PPI System registers**

| Register | Name |
|---|---|
| ICV_PPI_SACTIVER<n>_EL1 | Set interrupt Active |
| ICV_PPI_CACTIVER<n>_EL1 | Clear interrupt Active |
| ICV_PPI_ENABLER<n>_EL1 | Interrupt Enable and Disable |
| ICV_PPI_HMR<n>_EL1 | Interrupt Handling mode (Level or Edge) |
| ICV_PPI_SPENDR<n>_EL1 | Set interrupt Pending |
| ICV_PPI_CPENDR<n>_EL1 | Clear interrupt Pending |
| ICV_PPI_PRIORITYR<n>_EL1 | Interrupt Priority |

I_CBBCV To enable context switching, the configuration of virtual PPIs is also accessible using the following registers:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Table 2.16: Hypervisor configuration PPI System registers**

| Register | Name |
|---|---|
| ICH_PPI_ACTIVER<n>_EL2 | Interrupt Active, accesses same state as ICV_PPI_SACTIVER<n>_EL1 and ICV_PPI_CACTIVER<n>_EL1. |
| ICH_PPI_DVIR<n>_EL2 | Controls whether Pending physical PPIs are directly injected as virtual PPIs to the Virtual Interrupt Domain. |
| ICH_PPI_ENABLER<n>_EL2 | Alias of ICV_PPI_ENABLER<n>_EL1. |
| ICH_PPI_PENDR<n>_EL2 | Interrupt Pending, accesses same state as ICV_PPI_SPENDR<n>_EL1 and ICV_PPI_CPENDR<n>_EL1. |
| ICH_PPI_PRIORITYR<n>_EL2 | Alias of ICV_PPI_PRIORITYR<n>_EL1. |

$R_{PQDYK}$ If the Virtual CPU interface is supported, a virtual PPI is permitted to be a candidate HPPI if and only if all of the following are true:

- ICV_PPI_ENABLER<n>_EL1.EN is 1.
- ICV_PPI_SPENDR<n>_EL1.PEND is 1.
- ICV_PPI_SACTIVER<n>_EL1.ACTIVE is 0.

$R_{JMKJM}$ The CPU interface selects the virtual candidate HPPI among the virtual PPIs by selecting the highest priority PPI that meets the virtual candidate HPPI selection criteria.

$R_{ZXWQD}$ If more than one virtual PPI meet the virtual candidate HPPI selection criteria, it is IMPLEMENTATION DEFINED which of those PPIs becomes the virtual candidate HPPI.

$I_{SXKHV}$ A direct read or write to a PPI System register could cause an indirect write to its alias or a different PPI System register. The indirect write is only guaranteed to be visible to subsequent reads or writes if a Context synchronization event takes place after the direct read or write and before the subsequent reads or writes. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information.

$S_{KQQTX}$ When software at EL1 performs a direct write to an ICV_ register, this causes an indirect write to the corresponding ICH_ register. The value returned by a read at EL2 from the ICH_ register is guaranteed to observe the indirect write, only if there is a context synchronization event between the indirect write and the direct read. If FEAT_ExS is implemented and SCTLR_EL2.EIS is 0, an exception taken to EL2 is not a context synchronization event, and software at EL2 must issue an ISB before reading the ICH_ registers to observe the latest value written by a guest VM.

For example, ICH_PPI_PRIORITYR<n>_EL2 is an alias of ICV_PPI_PRIORITYR<n>_EL1. A direct write of a priority value for a PPI ID to ICV_PPI_PRIORITYR<n>_EL1 causes an indirect write of the priority value to ICH_PPI_PRIORITYR<n>_EL2 for the same PPI ID. A Context Synchronization Event is required after the write to ICV_PPI_PRIORITYR<n>_EL1 to ensure that a subsequent read of ICH_PPI_PRIORITYR<n>_EL2 returns the latest priority value.

### 2.10.1.1 Direct injection of virtual PPIs

$D_{QNZMN}$ The architecture supports *direct injection* of the physical PPI Pending state to the Pending state of a virtual PPI.

$I_{VRZWD}$ When ICH_PPI_DVIR<n>_EL2.DVI is 1, the Pending state of the corresponding physical PPI INTID is directly injected to the Pending state of a virtual PPI.

$R_{BMZFB}$ Unless a timer PPI is redirected under nested virtualization, when the Pending state of a physical PPI is directly injected to the Pending state of a virtual PPI, the INTID of the virtual PPI is the same as the INTID of the physical PPI.

See 2.10.1.2 *PPI redirection under nested virtualization* for more information about redirection of timer PPIs under nested virtualization.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{PGRWM}$ The Pending state of a virtual PPI is one of the following:

- If the corresponding physical PPI is not directly injected, the Pending state of the virtual PPI is determined by ICH_PPI_PENDR<n>.PEND<x>.
- If the corresponding physical PPI is directly injected, the Pending state of the virtual PPI is determined by ICC_PPI_xPENDR<n>.PEND<x>.

When the Pending state of a physical PPI is directly injected as the Pending state of a virtual PPI, the field in ICV_PPI_xPENDR<n>.PEND<x> corresponding to the virtual PPI becomes an alias of the field in ICC_PPI_xPENDR<n>.PEND<x> corresponding to the physical PPI, and the value of ICH_PPI_PENDR<n>.PEND<x> is IGNORED.

$I_{XXSHB}$ When a physical PPI is directly injected and becomes Pending, both the virtual and physical PPI are Pending and are permitted to be acknowledged in either the corresponding Physical Interrupt Domain or in the Virtual Interrupt Domain.

$S_{YDZLY}$ When a physical PPI is directly injected, Arm expects that the physical PPI is disabled to avoid separate interrupt handling routines attempting to service the same interrupt.

$I_{KKCLT}$ When the Pending state of a physical PPI is directly injected as the Pending state of a virtual PPI, it is the configuration of the virtual PPI that determines when the PPI is signaled and how the PPI is handled.

$I_{BKZQC}$ For example, if the Pending state of physical PPI 23 is directly injected as the Pending state of a virtual PPI, all of the following are true:

- The Pending state of physical PPI 23 is directly injected as the Pending state of virtual PPI 23.
- Virtual PPI 23 may be considered as a candidate HPPI in the Virtual Interrupt Domain when all of the following are true:
  - Physical PPI 23 is Pending.
  - ICV_PPI_ENABLER0_EL1.EN23 is 1.
  - ICV_PPI_xACTIVE0_EL1.ACTIVE23 is 0.
- ICV_PPI_PRIORITYR2_EL1.PRIORITY7 determines the Priority of virtual PPI 23.

$R_{JBKGB}$ If EL2 is not implemented, the Effective value of ICH_PPI_DVIR<n>_EL2.DVI is 0 for all implemented PPIs.

$I_{CCLDF}$ When the Pending state of a physical PPI is directly injected as the Pending state of a virtual PPI, all of the following are true.

- Writes to the corresponding field in ICH_PPI_PENDR<n> update the value of that field.
- Reads from the corresponding field in ICH_PPI_PENDR<n> return the value of that field.

The value of the corresponding field in ICH_PPI_PENDR<n> does not affect the value of the field corresponding to the virtual PPI in ICV_PPI_xPENDR<n>.

$S_{SVFBL}$ Hypervisor software may save and restore the Pending state of a directly injected Edge PPI by accessing to the fields corresponding to the physical PPI INTID in ICC_PPI_xPENDR<n>.

$I_{NVWGX}$ The Effective value of ICH_PPI_DVIR<n>_EL2.DVI is 0 if PPI <x> is not assigned to the Current Physical Interrupt Domain.

### 2.10.1.2 PPI redirection under nested virtualization

$R_{GYGNW}$ The outputs of the EL1 physical timer and EL1 virtual timer signal different PPI IDs when all of the following are true:

- FEAT_GCIE is implemented.
- HCR_EL2.{NV,NV1} is {1,0}.
- EL2 is implemented and enabled in the Security state selected by SCR_EL3.{NSE,NS}.

In this case, all of the following are true:

- When SCR_EL3.NS is 1, all of the following are true:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

  - If ICH_PPI_DVIR0_EL2.DVI30 is 1, all of the following are true:
    - Physical PPI 30 is directly injected as virtual PPI 26.
    - ICV_PPI_xPENDR0_EL1.PEND26 is an alias of ICC_PPI_PENDR_EL1.PEND30.
    - ICV_PPI_xPENDR0_EL1.PEND30 accesses the Pending state of virtual PPI 30.
    - ICH_PPI_PENDR0_EL2.PEND26 is IGNORED.
    - ICH_PPI_DVIR0_EL2.DVI26 is treated as 0 for all other purposes than a direct read of the field.
  - If ICH_PPI_DVIR0_EL2.DVI27 is 1, all of the following are true:
    - Physical PPI 27 is directly injected as virtual PPI 28.
    - ICV_PPI_xPENDR0_EL1.PEND28 is an alias of ICC_PPI_PENDR_EL1.PEND27.
    - ICV_PPI_xPENDR0_EL1.PEND27 accesses the Pending state of virtual PPI 27.
    - ICH_PPI_PENDR0_EL2.PEND28 is IGNORED.
    - ICH_PPI_DVIR0_EL2.DVI28 is treated as 0 for all other purposes than a direct read of the field.
- When SCR_EL3.NS is 0, all of the following are true:
  - If ICH_PPI_DVIR0_EL2.DVI30 is 1, all of the following are true:
    - Physical PPI 30 is directly injected as virtual PPI 20.
    - ICV_PPI_xPENDR0_EL1.PEND20 is an alias of ICC_PPI_PENDR_EL1.PEND30.
    - ICV_PPI_xPENDR0_EL1.PEND30 accesses the Pending state of virtual PPI 30.
    - ICH_PPI_PENDR0_EL2.PEND20 is IGNORED.
    - ICH_PPI_DVIR0_EL2.DVI20 is treated as 0 for all other purposes than a direct read of the field.
  - If ICH_PPI_DVIR0_EL2.DVI27 is 1, all of the following are true:
    - Physical PPI 27 is directly injected as virtual PPI 19.
    - ICV_PPI_xPENDR0_EL1.PEND19 is an alias of ICC_PPI_PENDR_EL1.PEND27.
    - ICV_PPI_xPENDR0_EL1.PEND27 accesses the Pending state of virtual PPI 27.
    - ICH_PPI_PENDR0_EL2.PEND19 is IGNORED.
    - ICH_PPI_DVIR0_EL2.DVI19 is treated as 0 for all other purposes than a direct read of the field.

$I_{QKGXM}$ All fields corresponding to implemented PPIs in ICH_PPI_PENDR0_EL2 access the Pending state of the corresponding virtual PPI, regardless of whether a directly injected PPI is being redirected to a different virtual INTID due to nested virtualization. This means that all of the following are true:

- When Physical PPI 30 is directly injected as virtual PPI 26, all of the following are true:
  - Whether virtual PPI 26 is considered Pending is determined by the Pending state of physical PPI 30.
  - An access to ICH_PPI_PENDR0_EL2.PEND26 accesses the Pending state of virtual PPI 26.
  - The value of ICH_PPI_PENDR0_EL2.PEND26 is IGNORED, meaning that it is not used in determining whether virtual PPI 26 is considered Pending.
- When Physical PPI 27 is directly injected as virtual PPI 28, all of the following are true:
  - Whether virtual PPI 28 is considered Pending is determined by the Pending state of physical PPI 27.
  - An access to ICH_PPI_PENDR0_EL2.PEND28 accesses the Pending state of virtual PPI 28.
  - The value of ICH_PPI_PENDR0_EL2.PEND28 is IGNORED, meaning that it is not used in determining whether virtual PPI 28 is considered Pending.
- When Physical PPI 30 is directly injected as virtual PPI 20, all of the following are true:
  - Whether virtual PPI 20 is considered Pending is determined by the Pending state of physical PPI 30.
  - An access to ICH_PPI_PENDR0_EL2.PEND20 accesses the Pending state of virtual PPI 20.
  - The value of ICH_PPI_PENDR0_EL2.PEND20 is IGNORED, meaning that it is not used in determining whether virtual PPI 20 is considered Pending.
- When Physical PPI 27 is directly injected as virtual PPI 19, all of the following are true:
  - Whether virtual PPI 19 is considered Pending is determined by the Pending state of physical PPI 27.
  - An access to ICH_PPI_PENDR0_EL2.PEND19 accesses the Pending state of virtual PPI 19.
  - The value of ICH_PPI_PENDR0_EL2.PEND19 is IGNORED, meaning that it is not used in determining whether virtual PPI 19 is considered Pending.

## 2.10.2 Virtual priority masking

$D_{BMLRF}$ The *Virtual Priority Mask* is the value stored in ICV_PCR_EL1.PRIORITY.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{DDQBK}$ See 9.1 *Synchronization requirements for GICv5 System registers* for more information about synchronization requirements for updates to the Virtual Priority Mask.

$D_{CZRGJ}$ The *virtual running priority* is defined as the highest active priority stored in ICV_APR_EL1.

$I_{DLHDR}$ The virtual running priority is reported in ICV_HAPR_EL1.PRIORITY.

$D_{WDGVW}$ A virtual interrupt has *Sufficient priority* to be signaled when all of the following are true:

- The Priority of the interrupt is higher than the virtual running priority.
- The Priority of the interrupt is equal to or higher than the Virtual Priority Mask.

$I_{GLZCB}$ ICV_HPPIR_EL1 reports if there is a virtual HPPI of Sufficient priority for the Virtual Interrupt Domain. If there is a virtual HPPI of Sufficient priority, it also reports the INTID of that HPPI.

$I_{MDDZR}$ The behavior of ICV_HPPIR_EL1 in GICv5 is different from the ICV_HPPIRx_EL1 registers in GICv3.

In GICv3, ICV_HPPIRx_EL1 return the highest priority interrupt for the corresponding interrupt group irrespective of whether the interrupt has sufficient priority to be signaled to the PE.

GICv5 changes this behavior such that ICV_HPPIR_EL1 returns the highest priority interrupt only if it has Sufficient priority to be signaled.

### 2.10.3 Virtual interrupt signaling

$I_{VDZVT}$ When the virtual CPU interface is not configured to use Legacy operation, the GICv5 CPU interface determines the virtual HPPI for the Virtual Interrupt Domain from the virtual cand[illegible]e HPPIs from the IRI and the virtual PPIs and performs the following actions:

1. Applies priority masking based on software configured virtual priority masks and the virtual running priority.
2. Determines if the virtual HPPI signals the virtual IRQ interrupt to the PE.
3. When the virtual CPU interface signals the virtual IRQ interrupt, it further determines whether the virtual IRQ interrupt is signaled with Superpriority.

When Legacy operation is enabled, the Legacy virtual CPU interface determines the virtual HPPI from the list registers and performs the following actions:

1. Applies interrupt grouping and priority masking based on software configured virtual priority masks and the virtual running priority.
2. Determines if the virtual HPPI signals the virtual IRQ interrupt or the virtual FIQ interrupt to the PE.
3. When the virtual CPU interface signals the virtual IRQ interrupt, it further determines whether the virtual IRQ interrupt should be signaled with Superpriority.

This process is illustrated in Figure 2.5.

See 2.10.1 *Legacy virtual CPU interface* for more information about the Legacy virtual CPU interface:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential




**Figure 2.5: Virtual interrupt determination**

$R_{ZKPVN}$ When EL2 is implemented and is enabled in the Security state identified by SCR_EL3.{NSE, NS} and Legacy operation is disabled, the GICv5 CPU interface determines whether there is an HPPI for the Virtual Interrupt Domain when all of the following are true:

- ICV_CR0_EL1.EN is 1.
- ICH_VCTLR_EL2.EN is 1.

The HPPI for the Virtual Interrupt Domain is determined by selecting the interrupt with the higher priority from the following:

- The candidate HPPI presented by the IRI for the resident VPE.
- The candidate HPPI selected among the virtual PPIs.

There is no HPPI for the Virtual Interrupt Domain if any of the following are true:

- ICV_CR0_EL1.EN is 0.
- ICH_VCTLR_EL2.EN is 0.
- There are no candidate HPPIs for the Virtual Interrupt Domain.

---

**Note**

The candidate HPPI presented by the IRI for the resident VPE is not considered when selecting the HPPI for the Virtual Interrupt Domain when ICH_CONTEXTR_EL2.IRICHPPIDIS is 1.

See 2.10.5 *Selecting the resident VPE* for more information about managing the resident VPE.

---

$R_{JKZLR}$ If EL2 is not implemented, the Effective value of ICH_VCTLR_EL2.EN is 0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{XCFJD}$ If the candidate HPPI presented by the IRI for the resident VPE has the same priority as the candidate HPPI selected among the virtual PPIs, it is IMPLEMENTATION DEFINED which of the two candidate HPPIs is selected as the HPPI for the Virtual Interrupt Domain.

$R_{CGYDN}$ When there is at least one candidate HPPI for the Virtual Interrupt Domain, the CPU interface determines the HPPI for the Virtual Interrupt Domain in finite time.

When there is a change to the candidate HPPIs due to updates to the configuration and state of the virtual PPIs, or due to the IRI presenting a new candidate HPPI for the resident VPE, the CPU interface redetermines the HPPI for the Virtual Interrupt Domain in finite time.

$R_{NGHYT}$ The GICv5 CPU interface signals the vIRQ interrupt when there is an HPPI of Sufficient priority for the Virtual Interrupt Domain.

$I_{NJPKJ}$ When Legacy operation is disabled, the GICv5 virtual CPU interface does not signal the vFIQ interrupt.

See 2.10.7 *Legacy virtual CPU interface* for more information.

$I_{BVMRK}$ The behavior of the Legacy virtual CPU interface including when virtual interrupts are signaled is described in [3].

### 2.10.4 Virtual non-maskable interrupts

$R_{XKMQC}$ When there is an HPPI of Sufficient priority for the Virtual Interrupt Domain, the vIRQ interrupt is signaled to the PE with *Superpriority*[1] when the priority of the HPPI is 0.

Otherwise, the virtual HPPI is not signaled with Superpriority.

$D_{WHVFD}$ The virtual HPPI is an *NMI* when all of the following are true:

- The interrupt is signaled with Superpriority
- SCTLR_EL1.NMI is 1.

$I_{NQSPW}$ If the priority of the virtual HPPI is 0 and SCTLR_EL1.NMI is 0, the virtual HPPI is not an NMI.

### 2.10.5 Selecting the resident VPE

$G_{CQBBT}$ Hypervisor software manages multiple VMs and VPEs and selects different VPEs to be resident on a PE over time to allow multiplexing VPEs on a single PE.

$G_{PBWDL}$ Separate VPEs from the same VM can be resident on separate PEs at the same time to support VMs with multiple VPEs running concurrently.

$D_{JRKYV}$ The resident VPE is specified by writing the corresponding VPE ID and VM ID to ICH_CONTEXTR_EL2.

Each VM has a separate VPE ID namespace which means that two VPEs with the same VPE ID for different VM IDs specify different VPEs.

$I_{MLMSJ}$ At any given time for each PE in the system, there is either one VPE resident or no VPE resident. It is not possible for multiple VPEs to be resident on the same PE at the same time, even if those VPEs belong to different Security states.

$I_{YSVWR}$ The VM ID used in this specification is not directly related to the VMSA VMID[1].

$I_{BJSFF}$ When ICH_CONTEXTR_EL2.V is 0, no VPE is resident and ICH_CONTEXTR_EL2.{VM,VPE} are IGNORED.

$R_{XVRCV}$ If EL2 is not implemented, the Effective value of ICH_CONTEXTR_EL2.V is 0.

$I_{WCHLY}$ ICH_CONTEXTR_EL2.IRICHPPIDIS allows software to specify whether the candidate HPPIs presented by the IRI is considered when selecting the HPPI for the Virtual Interrupt Domain.

GIC system instructions that operate in the virtual interrupt domain access the configuration and state of virtual LPIs and SPIs belonging to the resident VM, irrespective of the value of ICH_CONTEXTR_EL2.IRICHPPIDIS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$S_{BYBKL}$ Making a VPE resident with ICH_CONTEXTR_EL2.IRICHPPIDIS set to 1 allows software to emulate an environment where the virtual IRI is disabled by setting ICH_CONTEXTR_EL2.IRICHPPIDIS to 1 for all the VPEs that belong to the same disabled virtual IRI.

$R_{LHFXQ}$ On a write that sets ICH_CONTEXTR_EL2.V to 1, the VM and VPE are resolved in the Physical Interrupt Domain identified by SCR_EL3.{NSE,NS}.

$R_{CDHXC}$ If SCR_EL3.{NSE,NS} is set to a Reserved value, a write that sets ICH_CONTEXTR_EL2.V to 1 has the following CONSTRAINED UNPREDICTABLE behaviors:

- No VPE is treated as being resident.
- SCR_EL3.{NSE,NS} is treated as an UNKNOWN non-reserved value.

$I_{FDPDR}$ An access by a GIC System instruction to the state or configuration of a virtual LPI or SPI, when the Current Interrupt Domain is the Virtual Interrupt Domain, applies to the resident VM of that PE. Apart from `GIC VDPEND`, where the target VM is specified as an argument to the instruction.

$R_{RXMNN}$ Changing SCR_EL3.{NSE,NS} when there is a resident VPE is CONSTRAINED UNPREDICTABLE with a choice of the following:

- ICH_CONTEXTR_EL2.V is cleared to 0.
- The CPU interface behaves as if no VPE is resident, but the value of ICH_CONTEXTR_EL2 is unchanged.
  - If ICH_CONTEXTR_EL2.V is subsequently written to 1, ICH_CONTEXTR_EL2.DB is RES0.

$S_{XCDSV}$ When switching Security state, Arm expects that software at EL3 writes to ICH_CONTEXTR_EL2 to either make no VPE resident or make a VPE of the new Security state resident.

$I_{KWFRF}$ Following a write that sets ICH_CONTEXTR_EL2.V to 1, one of the following is true:

- The VPE becomes resident on the PE and ICH_CONTEXTR_EL2.F is set to 0.
- The VPE is not resident on the PE and ICH_CONTEXTR_EL2.F is set to 1.

$R_{NHCGW}$ While ICH_CONTEXTR_EL2.V is 1, if ICH_CONTEXTR_EL2.VM and ICH_CONTEXTR_EL2.VPE select an invalid VPE, all of the following are true:

- The IRI behaves as if no VPE is resident on the PE.
- When a write updates ICH_CONTEXTR_EL2.V from 1 to 0, any doorbell request is ignored.
- It is CONSTRAINED UNPREDICTABLE whether a read of ICH_CONTEXTR_EL2.V returns 1 or 0.

See also 2.10.6 *Requesting VPE doorbells*.

$R_{TXVLW}$ When ICH_CONTEXTR_EL2.V is 1, if the VPE selected by ICH_CONTEXTR_EL2.{VM,VPE} becomes invalid and a new VPE selected by the same values subsequently becomes valid, it is CONSTRAINED UNPREDICTABLE whether the IRI continues to behave as if no VPE is resident on the PE or as if the new VPE had been made resident.

$R_{LLMWL}$ The effect of setting ICH_CONTEXTR_EL2.V to 1 is not guaranteed to be visible until after a Context synchronization event. After the Context synchronization event, if any virtual interrupt from the IRI is visible by reading the CPU interface registers, it is for the new resident VPE.

$I_{BRXVV}$ ICH_CONTEXTR_EL2.V is only updated from 0 to 1 as the result of a direct System register write. Following such a write, there is an indirect System register write to ICH_CONTEXTR_EL2.F which is guaranteed to be visible after a Context synchronization event.

A read of ICH_CONTEXTR_EL2.V and ICH_CONTEXTR_EL2.F returns the value last written to the fields, by either a direct System register write or an indirect System register write.

$R_{LJVNC}$ The Context synchronization event after setting ICH_CONTEXTR_EL2.{V,IRICHPPIDIS} to {1,0} will not complete until the IRI has presented the candidate HPPI for the new resident VPE, or the IRI has confirmed there is currently no candidate HPPI for the new resident VPE.

$I_{LXZBK}$ When using GICv5 Stream Protocol, when the IRS sends the `SetResident_Ack()` packet it indicates that it has either forwarded the first virtual interrupt for the new resident VPE or that there is no candidate HPPI for the resident VPE.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{WDKXF}$ ICH_HPPIR_EL2 returns the HPPI for the Virtual Interrupt Domain.

If there is no resident VPE, or ICH_CONTEXTR_EL2.IRICHPPIDIS is 1, the value returned represents whether there is a candidate HPPI among the virtual PPIs.

If there is a resident VPE, the value returned represents whether there is a candidate HPPI among the virtual PPIs and the virtual candidate HPPI presented by the IRI for the resident VPE.

$R_{JSHCW}$ The effect of clearing ICH_CONTEXTR_EL2.V to 0, or setting ICH_CONTEXTR_EL2.IRICHPPIDIS to 1, is not guaranteed to be visible until after a Context synchronization event. After the Context synchronization event, no virtual interrupt from the IRI is visible via the CPU interface registers.

$R_{VFKHN}$ Changing the value of ICH_CONTEXTR_EL2.VM or ICH_CONTEXTR_EL2.VPE when ICH_CONTEXTR_EL2.V is 1 is CONSTRAINED UNPREDICTABLE with the following permitted behaviors:

- The write is ignored, for all other purposes than a direct read of the re
- The previously selected VPE is made not resident an the V identified by the new ICH_CONTEXTR_EL2.VM and ICH_CONTEXTR_EL2.VP alues is made sident.

$S_{QPFVS}$ To make a VPE no longer resident, software performs the follow g seq ce:

1. Write ICH_CONTEXTR_EL2, clearing ICH_CONTE TR_EL2.V fro l t .
   ICH_CONTEXTR_EL2.DB controls whether a d bell inte upt is req sted for the previously resident VPE.
2. Issue an ISB.

To make a VPE resident, software performs th ollowi sequenc

1. Write ICH_CONTEXTR_EL2, setting I _C L2.V from 0 to 1.
   1. The ICH_CONTEXTR_EL2. M and _CONTEXTR_EL2.VPE fields specify which VPE is being made resident.
   2. The Domain for the V is taken n SCK L3.{NSE,NS}.
2. Issue an ISB.
3. Optionally read ICH_ ONTE TR_EL2.F to check if the operation succeeded.

To change the resident PE, software m first ensure no VPE is resident, by performing the following sequence:

1. Write ICH_ ONTEXT _EL2, clearing ICH_CONTEXTR_EL2.V from 1 to 0.
2. Write ICH_ ONTE TR_EL2, setting ICH_CONTEXTR_EL2.V from 0 to 1 and selecting the new resident VPE.
3. Iss an B.
4. ptionall ead ICH_ ONTEXTR_EL2.F to check if the operation succeeded.

### 2.10.6 Reques VPE doorbells

$D_{GFNFW}$ When a resi t VPE is made non-resident, software can optionally request a *VPE doorbell*.

The VPE doorbell is a request for a physical interrupt to signal that there is a candidate HPPI for the VPE when the VPE is not resident on any PE.

$I_{QYLFV}$ When a VPE is made non-resident by writing ICH_CONTEXTR_EL2.V from 1 to 0, software uses ICH_CONTEXTR_EL2.DB to specify whether a VPE doorbell is requested.

$I_{RWSJW}$ When a VPE is made non-resident and ICH_CONTEXTR_EL2.DB is set to 0, no doorbell is requested and any previously requested doorbells are no longer requested.

$I_{GVGWJ}$ ICH_CONTEXTR_EL2.DBPM allows software to specify a *VPE Doorbell priority mask*. If a VPE doorbell is requested, the corresponding physical interrupt is only signaled if the candidate HPPI for the VPE is greater than or equal to the value written to ICH_CONTEXTR_EL2.DBPM.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{VNZRM}$ The mechanism to configure the interrupt used for the VPE doorbell is IMPLEMENTATION DEFINED.

When the GICv5 system architecture is used, the VPE doorbell configuration mechanism is described in 4.10.7 *VPE doorbells*.

### 2.10.7 Legacy virtual CPU interface

$G_{VXDMK}$ GICv5 enables the use of unmodified GICv3.3 compatible virtual machines.

$I_{BSYYP}$ GICv5 provides the optional FEAT_GCIE_LEGACY extension to support GICv3.3 VMs.

$D_{QHKVW}$ When ICH_VCTLR_EL2.V3 is 1, *Legacy operation* is enabled.

$R_{BFNZT}$ When FEAT_GCIE_LEGACY is not implemented, the Effective value of ICH_VCTLR_EL2.V3 is 0.

$D_{TPCQT}$ When Legacy operation is enabled, the virtual CPU interface functionality is referred to as the *Legacy virtual CPU interface*.

$I_{HKXPV}$ When FEAT_GCIE_LEGACY is implemented and Legacy operation is enabled, the Legacy virtual CPU interface is managed through the AArch64 virtualization control System registers described in *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4*[3].

$I_{JBZMP}$ When Legacy operation is disabled, the Effective value of ICH_HCR_EL2.En is 0 and the Legacy virtual CPU interface is disabled.

When Legacy operation is enabled, ICH_HCR_EL2.En determines whether the Legacy virtual CPU interface is disabled.

The effects of enabling and disabling the GICv3 Virtual CPU interface using ICH_HCR_EL2.En is described in [3].

$I_{TQKBH}$ When HCR_EL2.IMO is 1 and Legacy operation is enabled, the Legacy virtual CPU interface is used and all of the following are true:

- When executing at EL1, all of the following are true:
  - All GICv5 System instructions are UNDEFINED.
  - Accesses to GICv5 System registers are UNDEFINED.
  - Accesses to GICv3 Group 0, Group 1, and Common System registers are to Legacy virtual CPU interface ICV_* registers.
- Accesses to ICC_MCR_EL1 use the GICv3.3 version of the register.

See Chapter 8 *System instructions* and Chapter 9 *System registers* for more information.

$R_{YZMJH}$ When executing at Exception levels below EL2 and Legacy operation is enabled, the Effective value of HCR_EL2.IMO is determined by HCR_EL2.IMO:

- When HCR_EL2.IMO is 0, the Effective value of HCR_EL2.FMO is 0.
- When HCR_EL2.IMO is 1, the Effective value of HCR_EL2.FMO is 1.

This means that when Legacy operation is enabled and executing at EL1, software either interacts entirely with the GICv5 physical CPU interface or entirely with the Legacy virtual CPU interface.

$I_{PJWMZ}$ When Legacy operation is enabled, HCR_EL2.FMO enables GICv3.3 virtual Group 0 interrupts to be signaled as virtual FIQ interrupts.

$I_{NFJYV}$ FEAT_GCIE_LEGACY does not provide Interrupt bypass support as described in *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4*[3]. Therefore, when Legacy operation is enabled, the ICC_SRE_EL1.{DIB,DFB} field values are RAO/WI.

$I_{LDWTK}$ FEAT_GCIE_LEGACY only provides a System register interface to the Legacy virtual CPU interface. Therefore, when Legacy operation is enabled, the ICC_SRE_EL1.SRE field value is RAO/WI.

$R_{XXCLD}$ When Legacy operation is enabled, the ICV_CTLR_EL1.{ExtRange,RSS} field values are RAZ/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{CQLDH}$ When FEAT_GCIE_LEGACY is implemented, PPI 25 the GIC maintenance interrupt (GICMNT) is implemented. Otherwise, when FEAT_GCIE is implemented and FEAT_GCIE_LEGACY is not implemented, PPI 25 the GIC maintenance interrupt (GICMNT) is not implemented.

$I_{DQPYT}$ When FEAT_GCIE_LEGACY is implemented, if ICH_LR<n>_EL2.HW is set to 1, ICH_LR<n>_EL2.pINTID identifies a PPI in the Current Physical Interrupt Domain.

$I_{MTYHD}$ When FEAT_GCIE_LEGACY is implemented, ICH_VTR_EL2.SEIS and ICH_HCR_EL2.TSEI are RES0, meaning that local generation of system errors and trapping them to EL2 is not supported.

$I_{SHBPG}$ When FEAT_GCIE_LEGACY is implemented, ICH_VTR_EL2.PRIbits and ICH_VTR_EL2.PREbits are 0b100, meaning that 5 bits of priority and preemption are available to GICv3 VMs.

$I_{FPLXQ}$ GICv3 recorded the active priorities for Group 0 in ICH_AP0R<n>_EL2 and active priorities for Group 1 in ICH_AP1R<n>_EL2. In GICv5, the Legacy virtual CPU interface implements the same split, however the ICH_AP1R0_EL2.P<n> fields access the same state as the corresponding fields in ICH_APR_EL2.

See also:

- 9.2 *CPU interface registers*
- 9.6 *Hypervisor control registers*
- 9.7 *Legacy hypervisor control registers*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.11 GIC synchronous exception priorities

$I_{FTLLT}$ The relative priority of synchronous exceptions is described in R_ZFGJP of the L.b version of *Arm® Architecture Reference Manual, for A-profile architecture*[1].

$R_{MJLRP}$ Exceptions that occur as a result of attempting to execute an instruction that is UNDEFINED due to the configuration of ICH_VCTLR_EL2.V3 has a priority of 16.

$R_{KNQBS}$ For an exception taken to EL2 as the result of a configuration control in one of the following registers, the exception priority is 22:

- ICH_HCR_EL2.
- ICH_HFGITR_EL2.
- ICH_HFGRTR_EL2.
- ICH_HFGWTR_EL2.

This is the same priority as exceptions due to the other fine-grained t[illegible] registers.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 2.12 Interrupt ordering model and synchronization requirements

This section extends the Definitions of the Arm memory model in *Arm® Architecture Reference Manual, for A-profile architecture*[1].

**Interrupt Location**

An Interrupt Location comprises all state and configuration values relative to an interrupt. See 2.4 *Interrupt types and identifiers* for more information about interrupt state and configuration values. An Interrupt Location is uniquely identified by an INTID.

**Interrupt Effect**

GIC and GICR instructions might generate Interrupt Effects. An Interrupt Effect is an Interrupt Read Effect or an Interrupt Write Effect. Interrupt Effects can be either Explicit or Implicit. For example, a GIC instruction with the RCFG command generates an Explicit Interrupt Read Effect, a GICR instruction generates Implicit Interrupt Read Effects and might also generate an Implicit Interrupt Write Effect.

All Interrupt Write Effects are required to complete in finite time.

**Coherence order** There is a per-location *Coherence order* relation that provides a total order over all Interrupt Write Effects to that Location, starting with a notional Interrupt Write Effect of the initial value. The Coherence order of an Interrupt Location represents the order in which Interrupt Write Effects to the Interrupt Location arrive at their destination.

**Reads-from** The *Reads-from* relation couples Interrupt Read and Write Effects to the same Interrupt Location so that each Interrupt Read Effect is paired with exactly one Interrupt Write Effect in the execution of a program. An Interrupt Read Effect $E_2$ Reads-from an Interrupt Write Effect $E_1$, if and only if $E_1$ and $E_2$ are to the same Interrupt Location and $E_2$ takes its data from $E_1$.

For two Effects $E_1$ and $E_2$ if all of the following apply:

- $E_1$ is an Explicit Interrupt Read Effect generated by a GIC instruction with the RCFG command.
- $E_1$ appears in program order before $E_2$.
- $E_1$ and $E_2$ are to the same Interrupt Location.
- $E_2$ is an Interrupt Write Effect.

then it is architecturally forbidden that $E_1$ Reads-from $E_2$.

**Coherence-before, Coherence-after** An Interrupt Write Effect $E_1$ is *Coherence-before* an Interrupt Write Effect $E_2$ to the same Interrupt Location if $E_1$ is sequenced before $E_2$ in the Coherence order for the Interrupt Location.

An Interrupt Read Effect $E_1$ is *Coherence-before* an Interrupt Write Effect $E_2$ to the same Interrupt Location if $E_1$ Reads-from an Interrupt Write Effect $E_3$ and $E_3$ is Coherence-before $E_2$.

An Effect $E_2$ is *Coherence-after* an Effect $E_1$ if $E_1$ is *Coherence-before* $E_2$.

For two Effects $E_1$ and $E_2$ if all of the following apply:

- $E_1$ is an Implicit Interrupt Write Effect generated by a GICR instruction.
- $E_1$ appears in program order before $E_2$.
- $E_1$ and $E_2$ are to the same Interrupt Location.
- $E_2$ is an Interrupt Read Effect.

then it is architecturally forbidden that $E_2$ is Coherence-before $E_1$.

For two Effects $E_1$ and $E_2$ if all of the following apply:

- $E_1$ is an Implicit Interrupt Write Effect generated by a GIC instruction with the DI command.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- $E_1$ appears in program order before $E_2$.
- $E_1$ and $E_2$ are to the same Interrupt Location.
- One of the following applies:
  - $E_2$ is an Explicit Interrupt Read Effect generated by a `GIC` instruction with the `RCFG` command.
  - $E_2$ is an Implicit Interrupt Read Effect generated by a `GICR` instruction.

then it is architecturally forbidden that $E_2$ is Coherence-before $E_1$.

For two Effects $E_1$ and $E_2$ if all of the following apply:

- $E_1$ is an Explicit Interrupt Write Effect.
- $E_2$ is an Explicit Interrupt Read Effect.
- $E_1$ appears in program order before $E_2$.
- $E_1$ and $E_2$ are to the same Interrupt Location.

then it is architecturally forbidden that $E_2$ is Coherence-before $E_1$.

**Successful Read-Modify-Write pair of Interrupt Effects**

Two Effects $E_1$ and $E_2$ form a *Successful Read-Modify-Write pair* if and only if all of the following apply:

- $E_1$ is an Interrupt Read Effect.
- $E_2$ is an Interrupt Write Effect.
- $E_1$ and $E_2$ are to the same Interrupt Location.
- $E_1$ and $E_2$ are generated by the same instruction.

All Read-Modify-Write pairs of Interrupt Effects generated by `GIC` and `GICR` instructions are Successful Read-Modify-Write pairs of Interrupt Effects.

### 2.12.1 GIC and GICR ordering semantics

$I_{SYSPH}$ `GIC` instructions generate Interrupt Effects as required by the instruction's `command`. The `domain` together with the INTID in `<Xt>` specify the Interrupt Location of the interrupt Effect.

A `GIC` instruction with the `EOI` command does not generate any Interrupt Read Effects or Interrupt Write Effects.

A `GIC` instruction with the `RCFG` command generates an Interrupt Read Effect but no Interrupt Write Effects.

All other `GIC` instructions generate a Successful Read-Modify-Write pair of Interrupt Effects.

See *2.6 GIC System instructions* for more information about the A64 assembly language syntax for the GIC System instructions.

$R_{ZWSQL}$
- A `GIC` instruction with the `RCFG` command generates the following Effects:
  - A Register Read Effect $E_1$ of the `<Xt>` register.
  - An Explicit Interrupt Read Effect $E_2$ to the Interrupt Location determined by the `domain` and `<Xt>`.
  - An Indirect Register Write Effect $E_3$ to the System Register ICC_ICSR_EL1.

  In addition, all of the following apply:
  - There is an Intrinsic Data Dependency from $E_1$ to $E_2$.
  - There is an Intrinsic Data Dependency from $E_2$ to $E_3$.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{MLYCN}$ • A GIC instruction with the DI command generates the following Effects:
  – A Register Read Effect $E_1$ of the <Xt> register.
  – An Implicit Interrupt Read Effect $E_2$ to the Interrupt Location determined by the domain and <Xt>.
  – An Implicit Interrupt Write Effect $E_3$ to the Interrupt Location determined by the domain and <Xt>.

  In addition, all of the following apply:
  – There is an Intrinsic Data Dependency from $E_1$ to $E_2$.
  – There is an Intrinsic Data Dependency from $E_2$ to $E_3$.
  – $E_2$ and $E_3$ form a Successful Read-Modify-Write pair.

$R_{XHKDN}$ • A GIC instruction with a command other than RFCG, DI or EOI generates the following Effects:
  – A Register Read Effect $E_1$ of the <Xt> register.
  – An Explicit Interrupt Read Effect $E_2$ to the Interrupt Location determined by the domain and <Xt>.
  – An Explicit Interrupt Write Effect $E_3$ to the Interrupt Location determined by the domain and <Xt>.

  In addition, all of the following apply:
  – There is an Intrinsic Data Dependency from $E_1$ to $E_2$.
  – There is an Intrinsic Data Dependency from $E_2$ to $E_3$.
  – $E_2$ and $E_3$ form a Successful Read-Modify-Write pair.

$R_{FDSPZ}$ • A GICR instruction with the IA or NMIA command generates the following Effects:
  – An Indirect System Register Read Effect $E_1$ to ICC_HAPR_ELx.
  – Implicit Interrupt Read Effect $E_2$, $E_3$, ... $E_n$ to all n Interrupt Locations for the Current Interrupt Domain.
  – A Branching Effect $E_{n+1}$ that determines the HPPI.
  – If there is a HPPI and its priority is higher than the current running priority:
    * An Implicit Interrupt Write Effect $E_{n+2}$ to the Interrupt Location determined as the HPPI.
    * An Indirect System Register Write Effect $E_{n+3}$ to ICC_HAPR_ELx.
  – A Register Write Effect $E_{n+4}$ of the <Xt> register.

  In addition, all of the following apply:
  – There is an Intrinsic Data Dependency from $E_2$ to $E_{n+1}$.
  – There is an Intrinsic Data Dependency from $E_3$ to $E_{n+1}$.
  – ...
  – There is an Intrinsic Data Dependency from $E_n$ to $E_{n+1}$.
  – There is an Intrinsic Control Dependency from $E_{n+1}$ to $E_{n+4}$.
  – If there is an HPPI and $E_k$ is the Interrupt Read Effect from its Interrupt Location:
    * There is an Intrinsic Data Dependency from $E_k$ to $E_{n+2}$.
    * There is an Intrinsic Control Dependency from $E_{n+1}$ to $E_{n+2}$.
    * There is an Intrinsic Control Dependency from $E_{n+1}$ to $E_{n+3}$.
    * $E_k$ and $E_{n+2}$ form a successful Read-Modify-Write pair.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 2.12.2 GSB instruction semantics

$D_{KZFML}$ The A64 assembly language syntax for GIC synchronization barrier instructions is one of the following:

- `GSB SYS`
- `GSB ACK`

$D_{DLHHZ}$ A GSB instruction generates an Effect named after the instruction.

- A `GSB SYS` instruction generates a `GSB SYS` Effect.
- A `GSB ACK` instruction generates a `GSB ACK` Effect.

### 2.12.3 GIC Ordering Model

#### 2.12.3.1 GIC Ordering Relations

**GIC-hazard-ordered-before**

An Effect $E_1$ is GIC-hazard-ordered-before an Effect $E_2$ if and only if all the following apply:

- $E_1$ is an Explicit Interrupt Read Effect generated by a `GIC` instruction with the `RCFG` command.
- $E_1$ appears in program order before $E_3$.
- $E_1$ and $E_3$ are to the same Interrupt Location.
- $E_3$ is an Interrupt Read Effect.
- $E_3$ is Coherence-before $E_2$.
- $E_2$ is an Explicit Interrupt Write Effect.

If an Effect $E_1$ is GIC-hazard-ordered-before an Effect $E_2$ then $E_1$ is Hazard-ordered-before $E_2$.

**GSB-ordered-before** An Effect $E_1$ is GSB-ordered-before an Effect $E_2$ if and only if any of the following apply:

- All of the following apply:
  - $E_1$ is an Implicit Interrupt Effect generated by a `GICR` instruction.
  - $E_2$ is a GSB ACK Effect.
  - $E_1$ appears in program order before $E_3$.
  - $E_3$ appears in program order before $E_2$.
  - It is not the case that E2 is an Implicit Instruction Memory Read Effect.
- All of the following apply:
  - $E_1$ is an Interrupt Effect.
  - $E_3$ is a GSB SYS Effect.
  - $E_1$ appears in program order before $E_3$.
  - $E_3$ appears in program order before $E_2$.
  - It is not the case that E2 is an Implicit Instruction Memory Read Effect.

If an Effect $E_1$ is GSB-ordered-before an Effect $E_2$ then $E_1$ is Locally-ordered-before $E_2$.

If an Effect $E_1$ is GSB-ordered-before an Effect $E_2$, $E_2$ is an Instruction Fetch Barrier Effect and $E_2$ is in program-order before $E_3$ then $E_1$ is Instruction-fetch-barrier-ordered-before $E_3$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 2.12.3.2 Adaptations to existing Ordering relations

The following relations are defined in the chapter "Ordering requirements defined by the formal concurrency model" in *Arm® Architecture Reference Manual, for A-profile architecture*[1] and adapted to account for the Interrupt Read or Write Effects.

**Basic Dependency** There is a Basic dependency from an Effect E1 to an Effect E2 if and only if all the following apply:

- Any of the following applies:
  - E1 is an Explicit Memory Read Effect.
  - E1 is an Interrupt Read Effect.
  - E1 is a Register Read Effect.
- Any of the following applies:
  - There is a Dependency through registers and memory from E1 to E2.
  - E1 and E2 are the same Effect.

**Data dependency** There is a Data dependency from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Effect.
- There is a Basic dependency from $E_1$ to $E_3$.
- $E_3$ affects the data value written by $E_2$.
- There exists a chain of Intrinsic Data Dependency from $E_3$ to $E_2$.
- $E_2$ is an Explicit Memory Write Effect.
- It is not the case that $E_1$ and $E_2$ are generated by the same instruction.

**Address dependency**

There is an *Address dependency* from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Read Effect.
- There is a Basic dependency from $E_1$ to $E_3$.
- $E_3$ affects the address of the Location accessed by $E_2$.
- There exists a chain of Intrinsic data dependency from $E_3$ to $E_2$.
- Any of the following applies:
  - $E_2$ is an Explicit Memory Effect.
  - $E_2$ is an Implicit Tag Memory Read Effect.
  - $E_2$ is an Implicit TTD Memory Read Effect.
  - $E_2$ is a Hardware Update Effect.
  - $E_2$ is a TLBI Effect.
  - $E_2$ is a DC CVAU Effect.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- $E_2$ is an IC IVAU Effect.
- $E_2$ is an Interrupt Effect

- It is not the case that $E_1$ and $E_2$ are generated by the same instruction.

**Control dependency** There is a Control dependency from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Effect.
- There is a Basic dependency from $E_1$ to $E_3$.
- $E_3$ is a Conditional Branching Effect.
- $E_3$ appears in program order before $E_2$.

**Pick Basic dependency** There is a *Pick Basic dependency* from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is a Register Read Effect.
  - $E_1$ is an Interrupt Effect.
- Any of the following applies:
  - There is a Pick dependency through registers and memory from $E_1$ to $E_2$.
  - $E_1$ and $E_2$ are the same Effect.

**Pick Data dependency** There is a Pick Data dependency from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Read Effect.
- There is a Pick Basic dependency from $E_1$ to $E_3$.
- $E_3$ affects the data value written by $E_2$.
- Any of the following applies:
  - There is an Intrinsic Data Dependency from $E_3$ to $E_2$.
  - There is an Intrinsic Control Dependency from $E_3$ to $E_2$.
  - There exists a chain of Intrinsic Data Dependency or Intrinsic Control Dependency from $E_3$ to $E_2$.
- $E_2$ is an Explicit Memory Write Effect.
- It is not the case that $E_1$ and $E_2$ are generated by the same instruction.

**Pick Address dependency** There is a Pick Address dependency from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Read Effect.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- There is a Pick Basic dependency from $E_1$ to $E_3$.
- $E_3$ affects the address of the Location accessed by $E_2$.
- There exists a chain of Intrinsic Data Dependency from $E_3$ to $E_2$.
- Any of the following applies:
  - $E_2$ is an Explicit Memory Effect.
  - $E_2$ is an Implicit Tag Memory Read Effect.
  - $E_2$ is an Implicit TTD Memory Read Effect.
  - $E_2$ is a Hardware Update Effect.
  - $E_2$ is a TLBI Effect.
  - $E_2$ is a DC CVAU Effect.
  - $E_2$ is an IC IVAU Effect.
  - $E_2$ is an Interrupt Read Effect.
- It is not the case that $E_1$ and $E_2$ are generated by the same instruction.

**Pick Control dependency** There is a Pick Control dependency from an Effect $E_1$ to an Effect $E_2$ if and only if all the following apply:

- Any of the following applies:
  - $E_1$ is an Explicit Memory Read Effect.
  - $E_1$ is an Interrupt Read Effect.
- There is a Pick Basic dependency from $E_1$ to $E_3$.
- $E_3$ is a Conditional Branching Effect.
- $E_3$ appears in program order before $E_2$.

#### 2.12.3.3 GIC Observation Relations

**GIC-observed-by** An Effect $E_1$ is GIC-observed-by an Effect $E_2$ if and only if any of the following apply:

- All of the following apply:
  - $E_1$ is an Interrupt Write Effect.
  - $E_2$ Reads from $E_1$.
  - $E_2$ is an Interrupt Read Effect.
- All of the following apply:
  - $E_1$ is an Explicit Interrupt Effect.
  - $E_1$ is Coherence-before $E_2$.
  - $E_1$ and $E_2$ are from different Processing Elements.
  - $E_2$ is an Explicit Interrupt Write Effect.

If an Effect $E_1$ is GIC-observed-by an Effect $E_2$ then $E_1$ is Observed-by $E_2$.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 2.12.3.4 Ordering Requirements

**Atomicity of Successful Read-Modify-Write pair of Interrupt Effects**

For two Interrupt Effects $E_1$ and $E_2$ if and only if all the following apply:

- $E_1$ and $E_2$ form a successful Read-Modify-Write pair.
- There is an Interrupt Write Effect $E_3$.

then it is not the case that $E_1$ is Coherence-before $E_3$ and $E_3$ is Coherence-before $E_2$.

An architecturally well-formed execution must not exhibit a cycle in the Ordered-before relation as defined in the *Arm® Architecture Reference Manual, for A-profile architecture*[1] and extended by the definitions in this section GIC Ordering Model.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter 3
# GICv5 system architecture

RETIRED

This section defines the GICv5 system architecture, including its components and interrupt handling behavior.

$G_{BCPXN}$ The IRI manages LPIs and SPIs.

If the PEs in the system implement EL2, the IRI tracks which VMs and VPEs are defined and maintains their virtual interrupt state.

$I_{CKLJJ}$ This section describes an IRI implemented using the GICv5 system architecture.

$I_{YXYKQ}$ The GICv5 system architecture defines the following classes of system component:

1. The *Interrupt Routing Service* (IRS).
2. The *Interrupt Translation Service* (ITS).
3. The *Interrupt Wire Bridge* (IWB).

Figure 3.1 shows an overview of the GICv5 system architecture.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

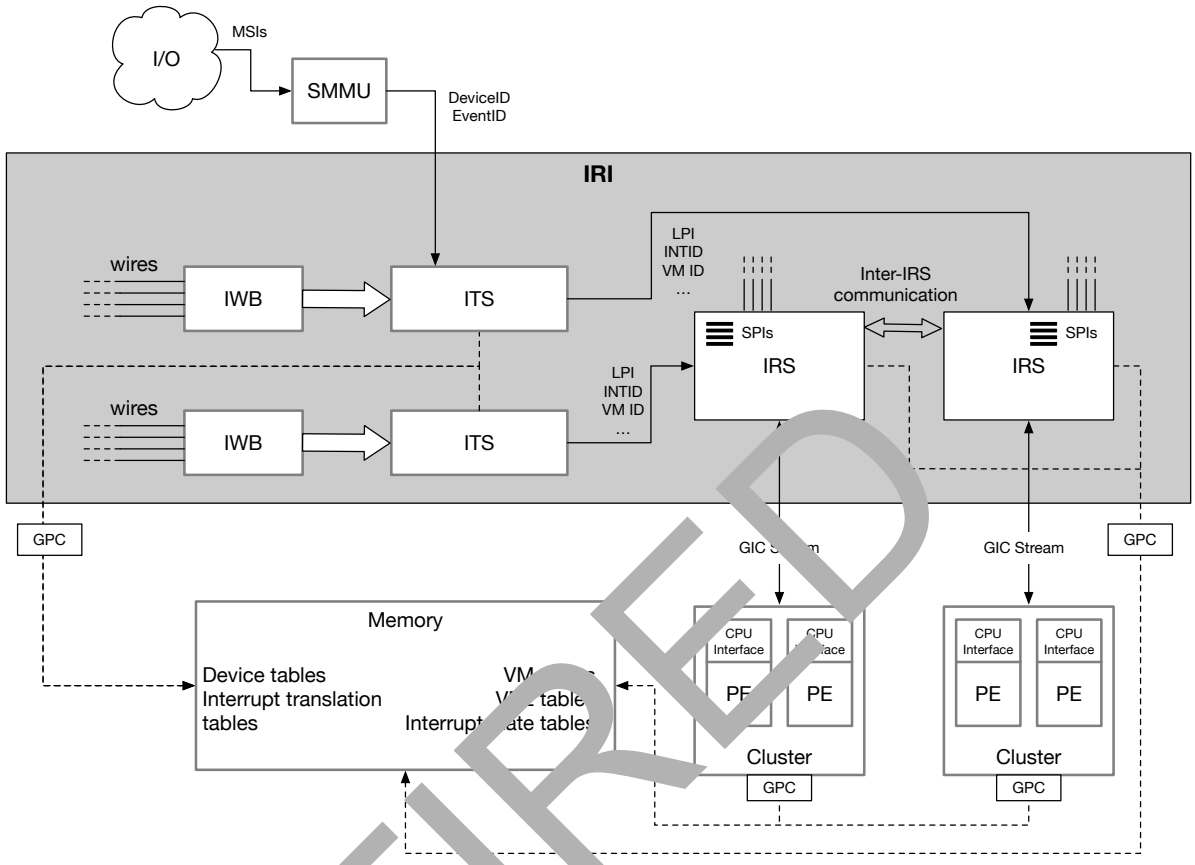



**Figure 3.1: System overview**

$D_{MZRCX}$ A GICv5 system using the GICv5 system architecture comprises the following:

- One or more IRSs.
- Zero or more ITSs.
- Zero or more IWBs.

$I_{BFXPB}$ A GICv5 system has optional support for GIC PMUs. See Chapter 7 *GIC Performance Monitoring Unit (PMU)* for more information.

$I_{ZWPSS}$ IRSs and ITSs rely on data structures stored in memory to manage the routing and translation of interrupts.

Software allocates memory for the data structures used by the IRSs and ITSs and configures these components using memory-mapped I/O interfaces.

$D_{QJPLJ}$ An IRS supports the following types of incoming interrupt events from peripherals:

**Shared Peripheral Interrupts (SPIs)** SPIs are used for interrupt signals that are directly connected to the IRS, for example, using directly wired connections or via an IMPLEMENTATION DEFINED mechanism. SPIs do not rely on memory for storage and may be used for interrupts that must meet certain early boot, reliability, or real-time guarantees. Whether SPIs provide any guarantees of fixed latency is a property of a given implementation. The number of supported SPIs is a fixed property of a given implementation.

**Logical Peripheral Interrupts (LPIs)** LPIs are used for interrupt signals that are translated by an ITS and forwarded to the IRS as well as for VPE doorbells. LPIs are managed by the IRSs in the system and the state and configuration of the LPIs are stored in memory allocated by system software.

Software may also signal interrupts managed by the IRS, for example to support IPIs and software emulated peripherals. See 2.5 *Inter-Processor Interrupts* and 2.6 *GIC System instructions* for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_DWZTM When an IRS detects or receives an interrupt event, it updates the Pending state of the LPI or SPI specified by the interrupt event INTID. An interrupt event can either set or clear the Pending state depending on the interrupt event type.

The IRS supports IPIs by allowing CPU Interfaces to send commands to the IRS to make an interrupt Pending.

The IRSs keep track of all Pending, Inactive, and Enabled interrupts and route them to the target PEs. An IRS forwards an interrupt to a PE if it meets the following criteria:

- The interrupt targets that PE, either directly or through 1ofN routing.
- The interrupt is Pending, Inactive, and Enabled.
- The interrupt has the highest priority among all interrupts that are Pending, Inactive, and Enabled for that PE.

A system can include one or more IRSs and an implementation manages shared interrupt state across IRSs. Each PE is connected to exactly one IRS and an IRS can be connected to multiple PEs. The GICv5 architecture does not define how multiple IRSs communicate with each other, for example in a multi-chip system. A system with multiple IRSs supports changing the affinity of an interrupt from a PE on one IRS to a PE on another IRS, without losing interrupt state or configuration.

The IRS also manages VM state, supports virtual interrupts, and manages VPE doorbells. The IRS uses data structures stored in memory to manage the state of interrupts and configuration of VMs and VPEs.

I_HNNWD An ITS provides translation of incoming interrupt events into LPI INTIDs and forwards the translation results as interrupt events for an IRS. Interrupt events can be generated from wired interrupt signals connected to an IWB or from Message Signaled Interrupts (MSIs) of other subsystems. For example, an ITS can translate MSIs generated by PCIe endpoints routed through an SMMU. It can also translate MSIs generated by other system-specific subsystems. ITS translations specify whether an event should be signaled as a physical interrupt or as a virtual interrupt that is injected directly into a VM. The ITS uses data structures stored in memory to perform translation of events.

I_HXBQP An IWB generates interrupt events from wired interrupt signals and forwards the events to an ITS. An IWB supports both edge-triggered and level-sensitive wires, and associates each wired interrupt with a Security state. The GICv5 architecture supports systems with zero to any number of IWBs.

I_WWTGL The GICv5 system architecture maintains the Arm CPU architecture security guarantees. As in the PE architecture, the IRSs and ITSs can support multiple Security states and access to memory respect that each location in memory belongs to a separate PAS. For a system that implements the Realm Management Extension (RME), this means all accesses are subject to Granule Protection Checks (GPCs). The GPC mechanism can be implemented in various ways and is outside the scope of this document. Even though the GPC is shown in Figure 3.1 as occurring downstream from the ITS and IRS, it is possible to leverage the SMMU GPC mechanism for this purpose. See *Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3*[4] for more information.

Memory-mapped accesses to the IWB, ITS, and IRS registers are also subject to PAS filtering[5]. PAS filtering For the IWB occurs at the level of individual registers. PAS filtering for the ITS and IRS is performed at page granularity, either internally or externally to the component.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 3.1 Interrupt Domains

I_LTWNQ In a GICv5 system, the supported Interrupt Domains depend on the Security states implemented by the PEs. See 2.3 *Interrupt Domains* for more information.

I_ZGKPP Each GICv5 system component may implement support for a subset of the Interrupt Domains supported by the PEs, except from the IRS, which implements support for the same set Interrupt Domains as the connected PEs. See 4.3 *IRS Domains* for more information.

**Note**

Implementing different Security state configurations across PEs in a system is not supported. This restriction exists for reasons outside the scope of this specification See [2] for mo tion.

R_VDTVW A Physical Interrupt Domain is associated with a Security state and a S:

| Phy al Interrupt Do in | Security state | PAS |
|---|---|---|
| Non cu | Non-secure | Non-secure |
| cure | Secure | Secure |
| alm | Realm | Realm |

When the EL3 Interrupt Domain i uppo d, it is her associated with the Secure or Root Security state and PAS as follows:

| Supported Security states | EL3 Interrupt Domain Security state | EL3 Interrupt Domain PAS |
|---|---|---|
| Non-secu , Secure | Secure | Secure |
| Non-sec , Re m, Root | Root | Root |
| N ecure, alm, Se e, Root | Root | Root |

I_KGBSN The I S as ciated w a Physical Interrupt Domain is used in the following scenarios:

- Whe memory-mapped registers of the IWB, ITS, and IRS are accessed, the PAS defines the Physical Interrup Domain of the registers being accessed.
- When the ITS and IRS access data structures stored in memory, the access is always associated with a Physical Interrupt Domain and uses the PAS associated with that Physical Interrupt Domain.

If the GIC performs a memory access which fails a GPC, it experiences an external abort. The behavior for each GIC component is described in Chapter 4 *Interrupt routing service (IRS)* and Chapter 5 *Interrupt translation service (ITS)*.

D_VJVNZ The GICv5 architecture defines the *most privileged Security state* (MPSS) and *most privileged PA space* (MPPAS) as follows:

| Supported Security states | MPSS | MPPAS |
|---|---|---|
| Non-secure | Non-secure | Non-secure |
| Secure | Secure | Secure |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| Supported Security states | MPSS | MPPAS |
|---|---|---|
| Non-secure, Secure | Secure | Secure |
| Non-secure, Realm, Root | Root | Root |
| Non-secure, Realm, Secure, Root | Root | Root |

See also:

- 2.3 *Interrupt Domains*
- Chapter 4 *Interrupt routing service (IRS)*
- Chapter 5 *Interrupt translation service (ITS)*
- Chapter 6 *Interrupt Wire Bridge (IWB)*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 3.2 Communication between GIC system components

$R_{ZFCFX}$ The mechanisms used for communication between GICv5 system components are IMPLEMENTATION DEFINED.

For example, the mechanism used by the ITS to send an interrupt message to the IRS is IMPLEMENTATION DEFINED.

$D_{ZFVZM}$ The architecture supports the following flows of events between GICv5 system components:

- An IWB can send events to an ITS.
- An ITS can send events to an IRS.

Each communication path involves a source and a destination.

When a source sends an event to the destination, the event is *Accepted* when the destination has sent a reply to the source confirming that it has received the event.

An Accepted event is not guaranteed to have been handled by the destination. This means that all of the following are true:

- If the destination is an ITS, an Accepted event is not guaranteed to have been translated by the ITS.
- If the destination is an IRS, an Accepted event is not guaranteed to have been processed by the IRS.

See also Chapter 3 *GICv5 system architecture*

$I_{VFJSX}$ An event is always Accepted by the destination regardless of whether the destination is enabled or disabled.

$I_{MNGPV}$ An example interconnect of two GICv5 system components is the *AMBA® AXI Protocol Specification*[6].

In this example, the source is a manager and the destination is a subordinate.

The subordinate has Accepted an event when it has sent a Write Response to the manager.

Further, in this example, an ITS would send a Write Response to the IWB when the ITS has Accepted an event from the IWB.

In this example, it is assumed that transactions are not Bufferable. This means that AxCACHE[0] is deasserted and therefore no intermediate send a Read or Write Response to the manager.

$I_{MZDPH}$ The GICv5 Stream Protocol defines an optional standard interface between an IRS and a CPU interface.

The GICV5 Stream Protocol defines a two-way communication mechanism that does not rely on the concept of Accepted as defined in this specification.

See Chapter A *GICv5 Stream Protocol overview* for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 3.3 Coherency considerations for GIC data structures

$D_{NCTBH}$ PEs and GIC system components communicate using *shared data structures*.

$I_{JPHKW}$ The GIC system components may be fully coherent with PE caches, IO-coherent, or not coherent.

Software must take actions to ensure that writes from the PE are visible to the GIC, and vice versa.

$S_{MXSNW}$ Arm expects that firmware data structures describe to the OS or hypervisor whether the GIC is cache-coherent with PEs and is in the same Inner Shareability domain.

$I_{QGRVW}$ Arm recommends that all GIC system components are in the same Inner Shareability domain as the PEs.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 4
# Interrupt routing service (IRS)

RETIRED

$I_{BRKFW}$ An IRS implements support for LPIs and SPIs. An IRS is either connected to one or more PEs, or it is stand-alone with no direct PE connections. A system can contain multiple IRSs.

The GICv5 architecture does not define the topological relationship between PEs and IRSs. Each PE is connected to exactly one IRS and an IRS can be connected to multiple PEs. An IRS can also be standalone and not be connected to any PE. This can be useful to support system topologies where an IRS and its associated ITS are located separately from PEs, allowing a more flexible routing of interrupts and power management of components close to the PE.

The mechanism by which multiple IRSs communicate are IMPLEMENTATION DEFINED. The IRS stores interrupt state and configuration in memory provisioned to the IRS by software. In a system with multiple IRSs, the Affinity of an interrupt in an Interrupt Domain can be changed from a PE connected to an IRS to another one connected to another IRS, without losing interrupt state or configuration, if both IRSs can access the provisioned memory for that Interrupt Domain.

To support interrupt signaling in scenarios where memory is unavailable, such as during early boot, power transitions, or RAS handling, the IRS supports SPIs.

SPIs do not require a data structure allocated by software in order to be signaled, or to be managed and handled by PEs. Every SPI is assigned to an Interrupt Domain and the IRS exposes a programming interface to configure which Interrupt Domain an SPI is assigned to.

The IRS also provides support for virtual interrupts and manages VMs and VPEs. The IRS supports storing interrupt state and configuration in memory-backed data structures.

An IRS is responsible for:

- Generating SET_EDGE, SET_LEVEL, and CLEAR interrupt events for SPIs as a result of interrupts connected to the IRS being signaled.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Receiving SET_EDGE, SET_LEVEL, and CLEAR interrupt events from ITSs for LPIs.
- Generating SET_EDGE LPI interrupt events as a result of writes to the SETLPI register, if implemented.
- Selecting the physical candidate HPPIs among LPIs and SPIs for each PE connected to the IRS.
- Selecting the virtual candidate HPPIs among virtual LPIs and virtual SPIs for each resident VPE on PEs connected to the IRS.
- Delivering virtual and physical HPPIs to the CPU interface.
- Processing Interrupt Effects from GIC System instructions executed on PEs.
- Managing residency state of VPEs on PEs, including:
  - Generating doorbell interrupts for non-resident VPEs.

$I_{NZXZQ}$ In this section, the term PE(s) refers to PE(s) connected to the IRS.

$I_{FZLBL}$ Arm strongly recommends that all PEs connected to a GIC are application PEs running a common system software stack.

The architecture permits the PEs connected to an IRS to be a combination of application PEs and non-application PEs, as long as all PEs implement a GICv5 compliant CPU interface.

However, the GIC architecture does not provide features to provide isolation or prevent interference between equally privileged host software running in the same Interrupt Domain. Therefore, if the PEs connected to an IRS are running separate system software stacks, a mechanism to manage the shared IRS and other GIC components must be supported by system software.

$I_{QBJVX}$ The IRSs replace parts of the functionality of the Redistributors and Distributors in GICv3.

$I_{GCWBM}$ The IRS manages LPIs and SPIs.

$R_{FMTPZ}$ When a system includes more than one IRS, each IRS supports the same number of interrupt ID bits for each interrupt type.

$R_{DRRHH}$ Each PE is associated with exactly one IRS. Zero or more PEs can be connected to a single IRS.

$D_{LJQSX}$ The IRS where an interrupt is managed is referred to as the *local IRS* for that interrupt.

Similarly, the IRS that a PE is connected to is referred to as the *local IRS* for that PE.

$D_{CHPRY}$ A system with more than one IRS is referred to as a *multi-IRS system*.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 4.1 Communication between the IRS and the CPU interface

$R_{HWCBT}$ The mechanism for communication between a CPU interface in a PE and the local IRS is IMPLEMENTATION DEFINED.

$I_{NTXCC}$ Arm recommends that an implementation uses the GICv5 Stream Protocol for communication between PEs and IRSs.

See Chapter A1 *GICv5 Stream Protocol overview* for more information.

$I_{LTXQF}$ Access to virtual interrupt configuration and state is made in the context of the resident VPE on the PE. When the PE switches the Current Physical Interrupt Domain, the PE and the IRS must ensure that software cannot access configuration and state for interrupts in the context of a VPE from the previous Physical Interrupt Domain. The GICv5 Stream Protocol avoids this by tagging all commands with the Physical Interrupt Domain.

$I_{SBMVW}$ The IRS manages the state and configuration of the LPIs and SPIs, and selects a candidate HPPI from among LPIs and SPIs for each Interrupt Domain for each connected PE. The PE selects the HPPI for each Interrupt Domain from the candidate HPPI and the PPIs managed in the PE.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.2 Signaling interrupts

$D_{YDNDD}$ An interrupt event is *processed* when the Interrupt Effects of the event are Ordered-before an Interrupt Read Effect to the Interrupt Location.

$D_{KLYXV}$ An interrupt is *signaled* to an IRS when one of the following events occurs:

- An interrupt connected to the IRS as an SPI is asserted or de-asserted and generates an interrupt event to change the Pending state of the SPI.
- An ITS associated with the IRS generates an interrupt event to change the Pending state of an LPI.
- A write to the IRS_SETLPI register occurs to set the Pending state of an LPI.
- The IRS *processes* Interrupt Effects of a GIC System instruction that sets the Pending state of an interrupt.

$R_{JVVSD}$ Updates to the Pending state of interrupts occur only in response to defined signaling events. Speculative updates are not permitted.

$R_{QHGBY}$ The mechanism for signaling SPI interrupt events to the IRS is IMPLEMENTATION DEFINED.

$R_{MJNXL}$ The mechanism for communication between an ITS and the IRS is IMPLEMENTATION DEFINED.

$R_{FNGRT}$ When multiple interrupt events of different types are generated for the same Interrupt Location, their Interrupt Effects are ordered in the same order in which the events are generated.

$R_{VGJBD}$ An interrupt event received from an ITS contains the following information:

- Interrupt Domain.
- Physical or virtual interrupt.
- VM ID for a virtual interrupt.
- LPI INTID.
- Event type: SET_EDGE, SET_LEVEL, or CLEAR.

$R_{ZHJMZ}$ If more than one mapping for an LPI exists, either in a single ITS or across multiple ITSs, such that both mappings are capable of generating events for the same LPI to one or more IRSs concurrently, all of the following are true for events with multiple mappings:

- The order in which the events are received by the IRSs is UNKNOWN.
- The order in which the events are processed by the IRSs is UNKNOWN.
- It is CONSTRAINED UNPREDICTABLE whether the IRS processes all the events or ignores some of the events.

$R_{DJDSY}$ The following sequence avoids the CONSTRAINED UNPREDICTABLE behavior caused by multiple concurrent ITS mappings to the same LPI:

1. For each IRS where a mapping to the LPI exists, the following sequence is performed:
   1. The mapping is removed.
   2. A synchronization request is issued.
   3. A synchronization request is issued to the connected IRS.
2. A new mapping can be created on an ITS to the LPI.

See also 5.2.4 *ITS synchronization requests* and 4.5 *IRS synchronization requests*.

$I_{JNQGW}$ An IRS supports the following interrupt events generated for SPIs or received from ITSs to update the state and configuration of an interrupt:

- SET_EDGE.
- SET_LEVEL.
- CLEAR.

An IRS supports the following interrupt events when processing Interrupt Effects of a GIC System instruction that sets the Pending state of an interrupt:

- SET.
- CLEAR.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

When a write to IRS_SETLPI occurs, the IRS generates a SET_EDGE event locally for the specified LPI.

$R_{HHKMN}$ When the IRS processes an interrupt event, there is an Interrupt Write Effect to the interrupt state as follows:

- For a SET event, the interrupt Pending state is set to Pending.
- For a SET_EDGE event, the interrupt Handling mode is set to Edge and the Pending state is set to Pending.
- For a SET_LEVEL event, the interrupt Handling mode is set to Level and the interrupt Pending state is set to Pending.
- For a CLEAR event, the interrupt Pending state is set to Idle.

$I_{DRNMD}$ As interrupts are asynchronous, the Interrupt Write Effects of an operation by a PE might be overwritten by a subsequent event received from the interrupt source.

For SPIs where the Trigger mode is level-sensitive and where an interrupt signal is physically connected to the SPI, updates to the Pending state from GIC System instruction executed on a PE are permitted to be ignored by the IRS, as the Pending state may be determined by the current state of signal.

$R_{QKYQM}$ When a mapping for an LPI exists and the mapping is for events generated by an IWB, and the events are for a wire that is configured as level-sensitive in the IWB, the IRS is permitted to generate a CLEAR event for the LPI when the ITS mapping is updated or when the wire is assigned to a different Interrupt Domain in the IWB.

$R_{DQTDV}$ The SET_EDGE and SET_LEVEL interrupt events generate the following Interrupt Write Effects:

- An Interrupt Write Effect $E_1$ that updates the Handling mode.
- An Interrupt Write Effect $E_2$ that updates the Pending state.

For these events, $E_1$ is Ordered-before $E_2$.

$I_{NFMBM}$ The IRS_SETLPI register can be used in a system that does not support virtualization to support MSIs without the use of an ITS.

A hypervisor can also emulate this architected functionality to allow a VM to program a device assigned to the VM to generate an MSI without requiring the hypervisor to emulate an ITS.

$R_{LCSNJ}$ If an interrupt event specifies an unreachable INTID at the IRS where it is signaled, the event has no effect on any interrupt.

$I_{XZDTX}$ If an interrupt event specifies an unreachable INTID, and software error reporting is supported, this is reported with an appropriate error code in IRS_SWERR_STATUSR.

See also:

- *2.1 The GICv5 CPU interface*
- Chapter 5 *Interrupt translation service (ITS)*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.3 IRS Domains

RTCWZB The IRSs support interrupt routing and delivery with separate configurations for each supported Interrupt Domain.

DLDYMZ The GICv5 architecture defines an *IRS Domain* that provides interrupt routing and management services for an Interrupt Domain. An IRS Domain comprises register state and interrupt configuration structures across all IRSs in the system. Each IRS Domain is configured independently from other IRS Domains. The IST and the VM table are shared between all IRSs for the same IRS Domain, but are not shared across IRS Domains. The data structures are accessed with the PAS associated with the IRS Domain.

IFGLKF Figure 4.1 shows an overview of the IRS Domains and the register frames for each IRS Domain for a single-IRS system.

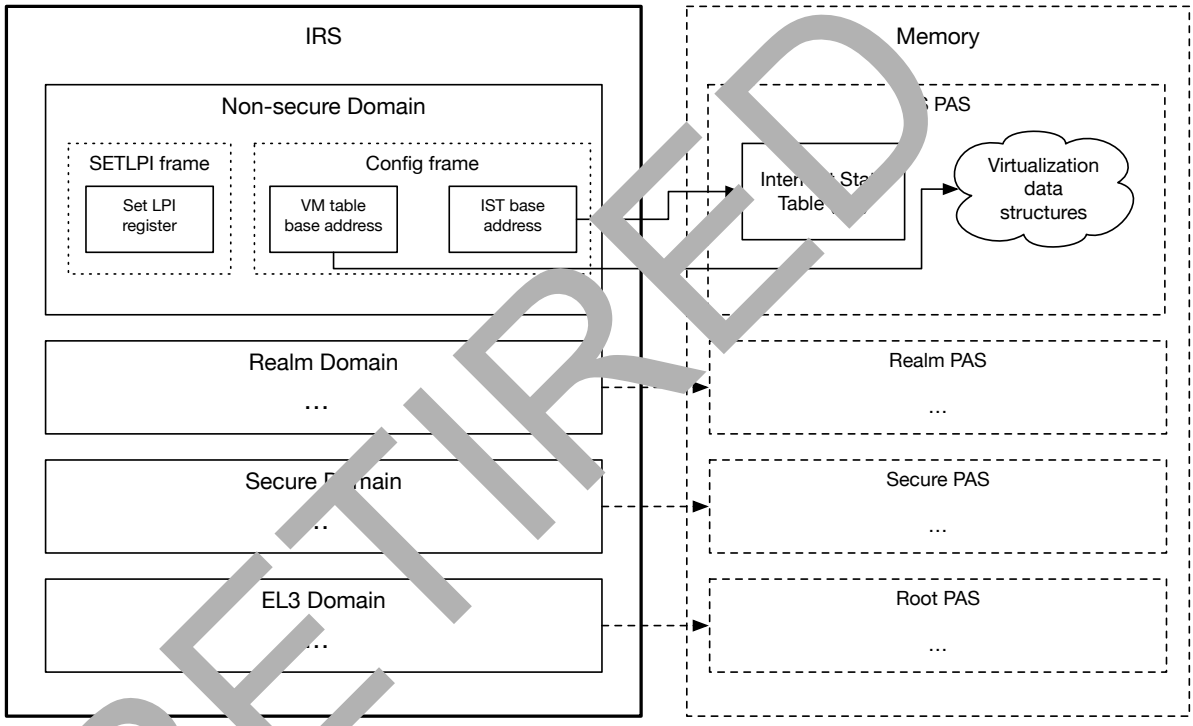



**Figure 4.1: IRS Domains in a single-IRS system.**

RGSNPJ Each IRS implements a separate IRS Domain for each Security state supported by the PEs in the system. Every PE connected to an IRS implements the same set of Security states.

See 3.1 *Interrupt Domains* for more information about the relationship between Security states and Interrupt Domains.

DWVCLJ The *most privileged IRS Domain* corresponding to the implemented IRS Domains as follows:

| Implemented IRS Domains | Most privileged IRS Domain |
|---|---|
| Non-secure | Non-secure |
| Secure | Secure |
| Non-secure, Secure, EL3 | EL3 |
| Non-secure, Secure, Realm, EL3 | EL3 |
| Non-secure, Realm, EL3 | EL3 |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{KZGKR}$ An IRS Domain is associated with the PAS of its corresponding Interrupt Domain.

$R_{XXWDT}$ An IRS exposes separate IRS configuration register frames for each IRS Domain.

$I_{CKHSF}$ The IRS Domain for an IRS configuration register frame is reported by IRS_IDR0.INT_DOM.

$I_{TCTJK}$ A VM is associated with exactly one IRS Domain which defines the Security state of the VM and its virtual interrupts.

Similarly, the VPEs in the VM are associated with the IRS Domain and Security state of the VM.

For example, a VPE in a Non-secure VM receives Non-secure virtual interrupts, and a VPE in a Realm VM receives Realm virtual interrupts.

$R_{NKVFJ}$ Each non-EL3 Physical Interrupt Domain has a separate namespace for Virtual Machines (VMs).

$R_{MHKVK}$ VMs are supported for the Secure, Realm, or Non-secure Interrupt Domains. VMs are not supported in the EL3 Interrupt Domain.

See also:

- 3.1 *Interrupt Domains*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.4 IRS Configuration

$R_{JJPMF}$ For each Interrupt Domain, each IRS exposes the following register frames:

- IRS_CONFIG_FRAME: An IRS configuration frame for settings that are common to all PEs connected to the IRS.
- IRS_SETLPI_FRAME: If IRS_IDR0.SETLPI is 1, a SETLPI register frame.

For each Interrupt Domain for each IRS, the IRS configuration register frame occupies a 64KB region in the system address space.

For each Interrupt Domain for each IRS, if IRS_IDR0.SETLPI is 1, the IRS SETLPI register frame occupies a 64KB region in the system address space.

Each register frame is 64KB-aligned.

See 10.2 *IRS register frames* for more information.

$I_{GVFTM}$ There is at most one SETLPI register frame for each IRS for an Interrupt Domain.

$I_{MZDWR}$ The SETLPI register frame is not implemented when support for physical LPIs is not implemented.

$S_{MTWNL}$ The base addresses of the IRS configuration frame and the IRS SETLPI register frame, if implemented, for each IRS and Interrupt Domain, are provided to software by firmware.

$R_{QHDWM}$ The value of IRS_IDR0.SETLPI is the same for all IRSs and all Interrupt Domains in the system.

$I_{NCXKF}$ Each IRS has a unique IRSID which is reported in IRS_IDR0.IRSID. The IRSID is unique for each IRS in the system. The IRSID reported in IRS_IDR0.IRSID is the same for each IRS_CONFIG_FRAME that belongs the same IRS.

$I_{NRPRB}$ The number of PEs connected to an IRS is reported by IRS_IDR1.PE_CNT.

$R_{FRRGD}$ Every PE is assigned a unique Interrupt Affinity ID across all IRSs in the system.

$R_{GSKQZ}$ The PE interrupt Affinity ID is the same across all implemented Interrupt Domains.

$R_{YBQML}$ If a system is reset with the same configuration, each PE has the same interrupt Affinity as before the reset.

$I_{XDFPV}$ If a system is reset and reconfigured in a way that presents a different system, for example by physically partitioning IRSs and PEs in a different topology after the system reset, each PE may be assigned a new interrupt Affinity ID in the new topology.

$I_{CWCZR}$ The PE interrupt Affinity ID is reported by the PE in ICC_IAFFIDR_EL1.

$I_{KZCQC}$ The PE interrupt Affinity ID is used to configure the target PE for Targeted interrupts.

It can also be used to discover which IRS a PE is connected to and vice versa.

$I_{LXNBN}$ Information about each PE connected to an IRS may be accessed by writing to IRS_PE_SELR to select a PE and accessing the following registers:

- IRS_PE_STATUSR.
- IRS_PE_CR0.

IRS_PE_STATUSR.IDLE reports whether the effects of a write to any of the above registers, including IRS_PE_SELR, have completed.

IRS_PE_STATUSR.V reports whether the value written to IRS_PE_SELR selects a valid PE connected to the IRS.

IRS_PE_CR0 provides access to configuration settings for the selected PE.

$S_{HHSMP}$ To configure a PE connected to an IRS, software performs the following sequence:

1. Software selects the PE by writing to IRS_PE_SELR.
2. Software polls IRS_PE_STATUSR.IDLE until it reads as 1 and ensures that IRS_PE_STATUSR.V is 1.
3. Software can access the configuration of a PE by accessing IRS_PE_CR0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

4. If software wrote to the PE configuration register, software polls IRS_PE_STATUSR.IDLE until it reads as 1 to ensure the effects of the write are complete.

$R_{KCZYL}$ When an IRS is reset, each PE connected to the IRS is reset as follows:

- If the IRS supports 1ofN interrupt routing, 1ofN PE selection is enabled for the PE.

$I_{PPXSP}$ The configuration of each PE at IRS reset is consistent with a read of IRS_PE_CR0.DPS returning 0 when the PE is selected in IRS_PE_SELR and there was no write to IRS_PE_CR0 since the reset.

$I_{MCXJN}$ Arm expects that firmware describes the association of PEs to an IRS to system software using firmware data structures.

The architecture also provides a mechanism for software to discover the association of PEs to an IRS which may be useful for system bring-up and debugging.

$S_{BCBQP}$ Software can discover the association of IRSs and PEs by following this sequence:

1. When a PE is powered on, software reads ICC_IAFFIDR_EL1 to obtain the Interrupt Affinity ID for that PE.
2. For each IRS, software writes the Interrupt Affinity ID for that PE to IRS_PE_SELR and performs the following actions:
   1. Poll IRS_PE_STATUSR.IDLE until it reads as 1.
   2. Read IRS_PE_STATUSR.V. If the value returned is 1, the IAFFID specifies a valid PE connected to that IRS.
   3. Otherwise, continue to the next IRS.

$I_{HTTRN}$ The number of supported IAFFID bits for physical interrupts is reported in IRS_IDR1.IAFFID_BITS.

$R_{GCGDD}$ The number of supported IAFFID bits for physical interrupts is the same across all IRSs and Interrupt Domains.

$I_{ZFSNT}$ The number of supported IAFFID bits is sufficient to uniquely identify all physical PEs in the system.

$I_{KVXQW}$ The number of supported VPE ID bits for virtual interrupts is reported in IRS_IDR4.VPE_ID_BITS.

$R_{NPTWH}$ When processing an Interrupt Write Effect generated by a GIC System instruction that updates the Affinity of an interrupt, all of the following are true:

- If the interrupt is a physical interrupt, unimplemented upper bits of IAFFID above IRS_IDR1.IAFFID_BITS are RES0.
- If the interrupt is a virtual interrupt, unimplemented upper bits of IAFFID above IRS_IDR4.VPE_ID_BITS are RES0.

See *4.9.2 The VPE table* for more information about the relationship between the IAFFID of a virtual interrupt and the VPE ID.

$R_{QQCXG}$ When IRS_CR0.IRSEN is 1 and IRS_CR0.IDLE is 1, an IRS is *enabled* for the Interrupt Domain and the IRS selects a candidate HPPI for each connected PE for the Interrupt Domain, following the rules in this chapter.

Otherwise, the IRS is *disabled* for the Interrupt Domain and the IRS does not select any candidate HPPI for the connected PEs for the Interrupt Domain.

$I_{ZYYLL}$ An IRS can access data structures to which it has a valid pointer, irrespective of whether the IRS is enabled for the corresponding Interrupt Domain.

$S_{SMLQY}$ To quiesce IRS accesses to IRS data structure, software makes the physical IST and VM table invalid, and polls the relevant status bits to ensure that all IRS memory accesses to these data structures have completed.

$R_{GQZNP}$ An IRS processes incoming interrupt events irrespective of the value of IRS_CR0.IRSEN.

$R_{LKCZP}$ An IRS processes Interrupt Effects generated by GIC System instructions, irrespective of the value of IRS_CR0.IRSEN.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{TQMVQ}$ Interrupts may need to be delivered to an IRS before the IRS has been enabled.

For example, an edge-triggered SPI may cause wake-up of an IRS that is in a low power mode and the signal should result in the Pending state of the SPI being updated as soon as the IRS is powered on.

See 4.11 *IRS power management* for more information.

See also:

- 2.6.2 *LPI and SPI configuration*
- 4.4.1 *Enabling and disabling the IRS*
- 4.6 *Interrupt configuration and state*
- 4.7 *The interrupt state table (IST)*
- 4.8 *Physical interrupts*
- 4.9 *Virtualization data structures*

### 4.4.1 Enabling and disabling the IRS

$R_{JHYGJ}$ When a write to IRS_CR0.IRSEN changes the value from 0 to 1, the IRS begins a transition from disabled to enabled for the Interrupt Domain.

The transition from disabled to enabled is complete when IRS_CR0.IDLE is 1.

The transition from disabled to enabled completes in finite time.

Enabling an IRS does not affect the state and configuration of LPIs.

$R_{SBXYL}$ When a write to IRS_CR0.IRSEN changes the value from 1 to 0, the IRS begins a transition from enabled to disabled for the Interrupt Domain.

The transition from enabled to disabled is complete when IRS_CR0.IDLE is 1.

The transition from enabled to disabled completes in finite time.

When the transition from enabled to disabled is complete, for each candidate HPPIs that were previously selected by the IRS for connected PEs for the Interrupt Domain while the IRS was enabled, one of the following is true:

- The candidate HPPI is acknowledged by the PE before the IRS is disabled.
- The candidate HPPI is not acknowledged and is no longer selected as the candidate HPPI for the PE.

$I_{ZMTMZ}$ The architecture allows enabling an IRS for an Interrupt Domain either before or after software provisions a valid physical LPI. See 4.8.1 *Physical LPIs* for more information.

See also:

- 4.7 *The interrupt state table (IST)*
- 4.9 *Virtualization data structures*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.5 IRS synchronization requests

$R_{HHDTJ}$ Software can request synchronization of interrupt events for the IRS Domain by writing 1 to IRS_SYNCR.SYNC.

Writing 1 to IRS_SYNCR.SYNC requests synchronization of interrupt events and ensures that the following events for the IRS Domain are processed:

- All interrupt events Accepted by the IRS Domain before the write to IRS_SYNCR.
- All interrupt events generated as a result of a write to an IRS_SETLPIR register before the write to IRS_SYNCR.
- All VPE doorbell events generated before the write to IRS_SYNCR.

Following a write to IRS_SYNCR.SYNC that requests synchronization, any Interrupt Write Effect $E_1$ to an Interrupt Location from a processed event is Ordered-before an Interrupt Read Effect $E_2$, when all of the following are true:

- There is a Memory Read Effect $E_3$ to IRS_SYNC_STATUSR that reads IRS_SYNC_STATUSR.IDLE as 1.
- $E_1$ is Ordered-before $E_3$.
- $E_3$ is Ordered-before $E_2$.

See 2.12 *Interrupt ordering model and synchronization requirements* for more information about observability of Interrupt Effects by PEs.

See 7.4 *Ordering and synchronization requirements for the GIC PMU* for more information about synchronization and counting events using the GIC PMU.

---

**Note**

An interrupt that meets the candidate HPPI condition after an interrupt event has been processed is not required to be acknowledged by an interrupt acknowledge instruction executed on a PE following an IRS synchronization request. This is because selecting the candidate HPPI is only required to happen in finite time. See 4.8.4 *Physical interrupt signaling* and 4.8.4 *Virtual interrupt signaling* for more information.

---

$I_{MJPSN}$ An interrupt event generates the following Interrupt Write Effects:

- If the interrupt event type is SET_EDGE or SET_LEVEL, an Interrupt Write Effect $E_1$ that updates the Pending state and an Interrupt Write Effect $E_2$ that updates the Handling mode.
- If the interrupt event type is SET or CLEAR, an Interrupt Write Effect that updates the Pending state.

$S_{BBHRQ}$ Software can use synchronization of events to ensure that interrupt events have been processed before repurposing an interrupt.

For example, if software disables a PCIe function, it can ensure that no additional Interrupt Effects originate from that function by performing the following sequence:

1. Ensure that all MSIs have completed from the PCIe function.
2. Issue a synchronization request to the ITS and poll for completion.
3. Issue a synchronization request to the IRS and poll for completion.

After this sequence, software can remove any old translations in the ITS and create new translations to the same INTID, knowing that Interrupt Effects to the INTID can only come from the new translations.

$I_{MXYLG}$ In a multi-IRS system, an IRS synchronization request applies to all IRSs in the system for the IRS Domain.

$R_{ZPNFL}$ If a write to IRS_SYNCR occurs when IRS_SYNC_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE whether the write synchronizes any interrupt events.

See also:

- 3.2 *Communication between GIC system components*
- Chapter 5 *Interrupt translation service (ITS)*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- 5.2.4 *ITS synchronization requests*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 4.6 Interrupt configuration and state

$I_{JVVTZ}$ The IRS manages the following states and configurations for each LPI and SPI:

- Pending.
- Active.
- Enabled.
- Handling mode.
- Priority.
- Routing mode.
- Affinity.

$I_{BWPPP}$ The IRS further manages the following for each SPI:

- Trigger mode.
- Interrupt Domain assignment.
- VM assignment.

$D_{HJCLV}$ An INTID specifying an LPI or SPI is *reachable* if the INTID is within the configured range for the interrupt type and if the IRS can access its configuration and state. Otherwise, the INTID is *unreachable*.

$R_{PGDXN}$ When the PE executes any of the following GIC System instructions to request the configuration of an unreachable INTID, the IRS indicates a failure in the response to the PE:

- `GIC CDRCFG`.
- `GIC VDRCFG`.
- `GIC LDRCFG`.

When the PE executes any other GIC System instruction that accesses the state and configuration of an unreachable INTID, there are no Interrupt Effects generated as a result of executing the instruction.

$I_{SRPZD}$ The maximum number of INTID bits supported by an IRS is reported by IRS_IDR2.ID_BITS.

$R_{SVDYQ}$ The value of IRS_IDR2.ID_BITS is the same for all IRSs for all Interrupt Domains in the system.

$I_{YKSNC}$ Arm recommends that an IRS supports a number of INTID.ID bits greater than or equal to the number of INTID.ID bits supported by the PEs connected to the IRS.

$R_{BBZWX}$ If an IRS supports fewer INTID.ID bits than a connected PE and the PE that specifies an INTID.ID beyond the INTID.ID range supported by the IRS, the INTID specifies an unreachable interrupt.

See also 2.6.2 *LPI and SPI configuration*.

$I_{GTVLX}$ Support for physical LPIs is optional. Whether the IRS supports physical LPIs is reported in IRS_IDR2.LPI.

Virtual LPIs are always supported when virtualization is supported and the value reported in IRS_IDR2.LPI has no impact on the support for virtual LPIs.

$I_{BNDTT}$ Arm recommends that IRSs in a system designed to run standard operating systems implements support for physical LPIs. If physical LPIs are not implemented, sufficient physical SPIs that are not connected to any interrupt sources must be implemented to be used for IPIs.

$I_{YKQDJ}$ Arm expects that support for LPIs is required for GICv5 systems in a future version of the *Arm® Base System Architecture 1.0C*[2].

$R_{MGQGH}$ When the IRSs do not implement support for LPIs, there are no ITSs or IWBs in the system.

$R_{RCGPT}$ The value of IRS_IDR2.LPI is the same for all IRSs for all Interrupt Domains in the system.

$I_{ZKLJR}$ The minimum number of LPI INTID.ID bits supported by an IRS is reported in IRS_IDR2.MIN_LPI_ID_BITS.

Arm expects that most hardware implementations will not require a minimum number of LPI INTID.ID bits. However, specifying a minimum number of LPI INTID.ID bits is useful for virtualization.

$R_{YHQLW}$ The value of IRS_IDR2.MIN_LPI_ID_BITS is the same for all IRSs for all Interrupt Domains in the system.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{PQBVF}$ The definition of reachable and unreachable interrupts applies to both physical and virtual interrupts.

The conditions for when a physical interrupt is reachable differ from when a virtual interrupt is reachable.

See 4.8.1 *Physical LPIs* and 4.8.2 *Physical SPIs* for more information about when physical interrupts are reachable.

See 4.10 *Virtual interrupts* for more information about when virtual interrupts are reachable.

$R_{ZMRVR}$ The Handling mode property of an interrupt controls whether the interrupt should be handled using edge-triggered or level-sensitive semantics as follows:

- When the interrupt Handling mode is Edge, the IRS clears the Pending state when the interrupt is acknowledged by the PE.
- When the interrupt Handling mode is Level, the IRS does not clear the Pending state when the interrupt is acknowledged by the PE.

$I_{XBVHV}$ The Priority property stores the Priority of an Interrupt.

$I_{XWHLD}$ The number of implemented Priority bits in the IRS is reported by IRS_IDR1.PRI_BITS and the minimum is 1 bit.

The number of Priority bits supported may be different across Interrupt Domains.

$I_{FRYMQ}$ Some operating systems have requirements for a minimum number of interrupt priority levels. These requirements will be captured as part of the *Arm® Base System Architecture SOC*[2].

$R_{TZYHS}$ For each Interrupt Domain, the number of supported Priority bits for the Interrupt Domain is the same across all IRSs in the system.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.7 The interrupt state table (IST)

$I_{LQMKQ}$ An *Interrupt State Table* (IST) is used by the IRS to store interrupt state and configuration.

$D_{PDQBB}$ An IST uses either a linear or 2-level structure.

For a linear IST, all of the following are true:

- The level 1 IST is omitted.
- The IST consists of a single level 2 IST.

$D_{XTNLJ}$ An IST is used to manage the state and configuration of interrupts in each of the following *IST contexts*:

- Physical LPI IST context: Used to manage physical LPIs for each Interrupt Domain. See 4.8.1 *Physical LPIs* for more information.
- Virtual LPI IST context: Used to manage virtual LPIs for VMs. See *A...ual LPIs* for more information.
- Virtual SPI IST context: Used to manage virtual SPIs for VMs. Se... 4.10.2 *Virt...l SPIs* for more information.

$D_{VBJJZ}$ For a linear IST, the base address of the IST specifies the location of t...evel 2 IST.

For a 2-level IST, the base address of the IST specifies the lo...ion of the le...l 1 IST.

Figure 4.2 shows an overview of the linear and 2-level s...cture.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

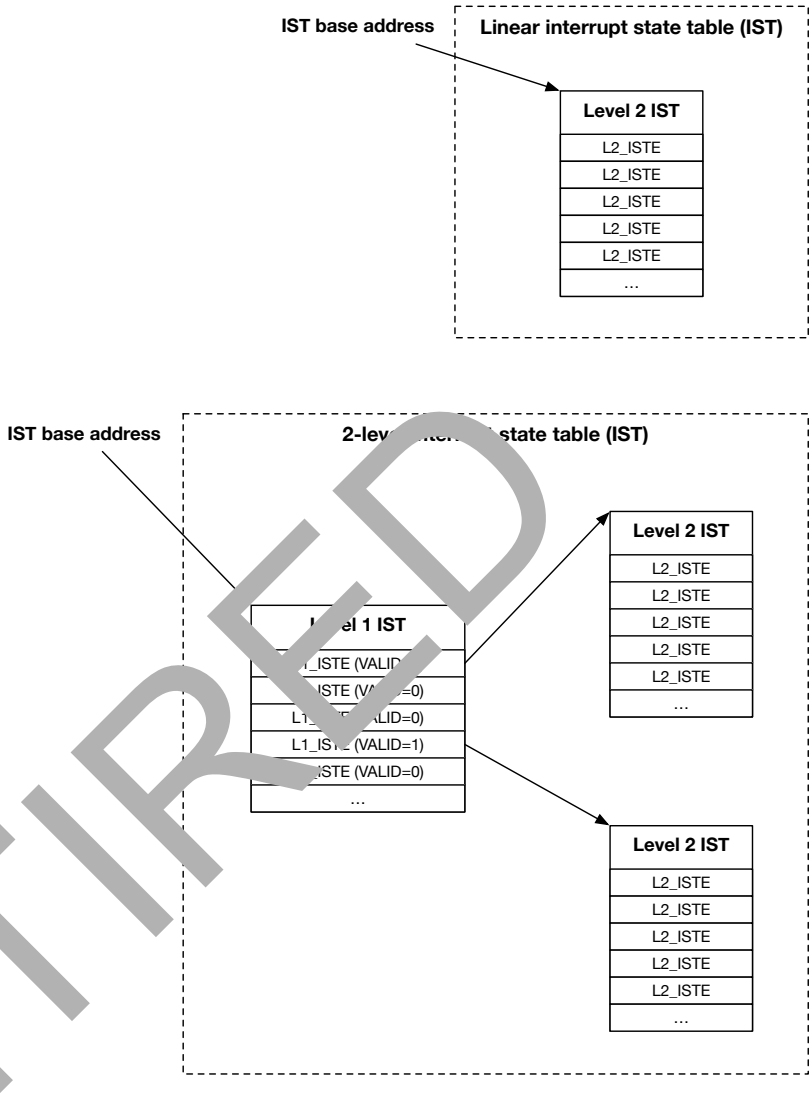



**Figure 4.2: IST structure overview.**

RETIRED

$S_{NWLKP}$ Software is responsible for allocating the IST from memory in the PAS associated with the Interrupt Domain where the IST is used.

$I_{MWLZJ}$ An IST is indexed using the INTID.ID for an interrupt within the configured range for the IST.

$D_{BXYVH}$ An IST is either considered *valid* or *invalid*.

The mechanism that determines whether the IST is valid or invalid depends on the IST context used.

When the IST transitions from being invalid to being valid, the IST *becomes valid*.

When the IST transitions from being valid to being invalid, the IST *becomes invalid*.

$I_{VVKXF}$ An IST is either valid or invalid across all IRSs.

$R_{SCLJY}$ When an IST is valid, the IRS Domain is permitted to access the IST for any reason, including speculative reads.

When an IST is invalid, the IRS Domain does not access the IST.

$D_{XQVBD}$ An IST stores a number of INTIDs for a single INTID.Type.

The mechanism that determines the number of INTIDs stored in an IST depends on the IST context used.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

IXTPFF An IST is used to manage the state and configuration for each INTID in the configured interrupt range for the IST context used.

INQBVH The configuration of an IST is controlled by the following parameters:

**STRUCTURE** Selects if the IST uses a linear or 2-level structure.

**ID_BITS** Selects how many INTIDs are stored in the IST.

The number of INTIDs stored in the IST determines the size of the single level 2 IST when using a linear table and the size of the level 1 IST when using a 2-level table.

The number of INTIDs stored in the IST also limits the supported size of each level 2 IST entry when the IRS stores metadata in the IST.

**L2SZ** For a 2-level IST, selects the size of each level 2 IST. The number of INTIDs stored in each level 2 IST depends on the size of a level 2 IST entry.

Arm recommends a linear IST is used when the total number of INTIDs can be stored in a single level 2 IST.

**Level 2 IST entry size** Selects the size of each level 2 IST entry.

The supported entry sizes depend on the number of INTIDs stored in the IST as well as the IRS requirements for storing metadata in the IST. See 4.7.4 *IST metadata* for more information.

These parameters are programmed in a way that is specific to the IST context used. For the physical IST context, each parameter is programmed in the physical IST base and configuration registers. For the virtual IST context, each parameter is programmed in the VM table entry.

See 4.8.1 *Physical LPIs*, 4.10.1 *Virtual LPIs*, and 4.10.2 *Virtual SPIs* for more information.

DTWFKY The size of the IST is calculated based on the number of INTIDs and the size of each entry, including any metadata if implemented, as follows:

- For a linear IST, the size of the level 2 IST is (Number of INTIDs) * (Level 2 IST entry size).
- For a 2-level IST, the size of the level 1 IST is ((Number of INTIDs) / (Level 2 IST size / Level 2 IST entry size)) * (Level 1 IST entry size).

IQBVSN IRS_IDR2.IST_LEVELS reports whether 2-level IST support is implemented.

RZGTCN The value of IRS_IDR2.IST_LEVELS is the same for all IRSs for all Interrupt Domains in the system.

INCSJH For a 2-level IST, an implementation may only implement support for a subset of the possible L2SZ values. The supported L2SZ values are reported by IRS_IDR2.IST_L2SZ.

RHVLTJ The value of IRS_IDR2.IST_L2SZ is the same for all IRSs for all Interrupt Domains in the system.

IPSTQD Arm strongly recommends that an implementation implements the L2SZ values corresponding to the stage 1 and stage 2 translation granule sizes in the PEs in the same system.

See also:

- 4.8.1 *Physical LPIs*
- 4.10.1 *Virtual LPIs*
- 4.10.2 *Virtual SPIs*

### 4.7.1 Level 2 IST management

DHMZZP A level 2 IST is considered *valid*, if the IST is valid and one of the following is true:

- The IST uses a linear structure.
- The IST uses a 2-level structure and the corresponding L1_ISTE.VALID is 1.

Otherwise, the level 2 IST is considered *invalid*.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{LWRCC}$ A level 2 IST *becomes valid* in any of the following scenarios:

- The IST uses a linear structure and the IST becomes valid.
- The IST uses a 2-level structure and any of the following occur:
  - A write to the level 2 IST map register for the IST context makes the level 2 IST valid.
  - The IST becomes valid and the corresponding L1_ISTE.VALID is 1.

$I_{XLXLR}$ The level 2 IST map register to use depends on the IST context used.

See 4.8.1 *Physical LPIs*, 4.10.1 *Virtual LPIs*, and 4.10.2 *Virtual SPIs* for more information.

$I_{ZDJBW}$ When a 2-level IST becomes valid, if L1_ISTE.VALID is 1 for more than one level 1 IST entry, more than one level 2 IST may become valid at the same time.

$R_{CWRMM}$ When the IST is valid and uses a 2-level structure, if software writes to an invalid level 1 IST entry and sets L1_ISTE.VALID to 1, the behavior of the IRS Domain is UNPREDICTABL

The UNPREDICTABLE behavior must not result in access to memory o ide the PAS sociated with the Interrupt Domain or in an access to memory that does not follow the behaviors scribed in 4.1 *IRS memory access rules*.

$R_{LFYMV}$ If software performs a write to any field in a valid level 1 IST e y, the IR Domain b avior is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access t emory side th s associated with the Interrupt Domain or in an access to memory that does not follow th ehavi s described in 4.12 *IRS memory access rules*.

$R_{DDPXH}$ To recover from UNPREDICTABLE behavior resul from a ncorrect rite to a level 1 IST entry, the IST is made invalid.

$I_{FSYTK}$ The architecture does not support making a ndividu l 2 IST invalid when using a 2-level structure for an IST.

$R_{FKYJV}$ When a write to the level 2 IST ma regist for the T context makes the corresponding level 2 IST valid, all of the following are true:

- The IRS generates the llov g Effects:
  - A Register Write Effect $E_1$ the status register for the IST context which sets the IDLE field to 0.
  - A Memory rite Effect $E_2$ the Location corresponding to the level 1 IST entry which updates L1_ISTE ALID from 0 to 1.
  - A Reg er Write ffect $E_3$ to the status register for the IST context which sets the IDLE field to 1.
- There is an I nsi order de ndency from the Register Write Effect $E_1$ to the Memory Write Effect $E_2$.
- Ther Intri s order pendency from the Memory Write Effect $E_2$ to the Register Write Effect $E_3$.

This eans tha f there is emory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_2$ is Ord d-befo E

$I_{LNDTY}$ When t T is valid and uses a 2-level structure, a level 2 IST is made valid by the following sequence:

1. Softwa rites the address of the new level 2 IST to L1_ISTE.L2_ADDR, with L1_ISTE.VALID remaining to be 0.
2. Software writes to the level 2 IST map register for the IST context.
3. The IRS performs a write to the level 1 IST entry that updates L1_ISTE.VALID from 0 to 1.
4. The IRS reports that the effects of the write to the level 2 IST map register for the IST context are complete.

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

$R_{SDVFP}$ When the IST is valid and uses a 2-level structure, if software writes to the level 1 IST entry when the status register IDLE field is 0 for the IST context, it is CONSTRAINED UNPREDICTABLE whether the IRS writes back the entry using the updated or old level 1 IST entry value.

The CONSTRAINED UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{XCRJV}$ A level 2 IST *becomes invalid* when the IST becomes invalid.

### 4.7.2 Initialization of level 2 IST entries

$R_{LXJHZ}$ When a level 2 IST becomes valid, the behavior is UNPREDICTABLE if all of the following are true:

- IRS_IDR2.ISTMD is 1.
- The metadata area of one or more level 2 IST entries are non-zero. See 4.7.4 *IST metadata* for more information about the metadata area.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{XJJYH}$ To recover from UNPREDICTABLE behavior resulting from a level 2 IST containing non-zero metadata becoming valid, the IST is made invalid.

$I_{BNPSN}$ A level 2 IST with non-zero metadata may become valid as part of an IMPLEMENTATION DEFINED IRS restore mechanism. See 4.11 *IRS power management* and 4.7.4 *IST metadata* for more information.

$R_{ZFBKH}$ When a level 2 IST becomes valid and L2_ISTE.HWU is not 0b0 for one or more level 2 IST entries, the behavior is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{XZWJX}$ To recover from UNPREDICTABLE behavior resulting from a level 2 IST containing entries with a non-zero HWU field becoming valid, the IST is made invalid.

$I_{GXNWY}$ A level 2 IST with non-zero HWU values may be restored valid as part of an IMPLEMENTATION DEFINED IRS restore mechanism. See 4.11 *IRS power management* for more information.

$R_{GXLJR}$ When a level 2 IST becomes valid and L2_ISTE.Pending is 1 for one or more level 2 IST entries, it is CONSTRAINED UNPREDICTABLE whether the vPEID is selected as the candidate HPPI even though it meets the candidate HPPI conditions.

$I_{CLSGT}$ If a level 2 IST containing entries with a non-zero Pending field becomes valid, once the corresponding interrupts are made Pending as a result of an interrupt being signaled, the IRS considers the interrupt when selecting a candidate HPPI.

$I_{XBHGH}$ See 4.8.4 *Physical interrupt signaling* and 4.10.4 *Virtual interrupt signaling* for more information about the candidate HPPI conditions.

$I_{HMFBP}$ A level 2 IST containing Pending interrupts may become valid as part of an IMPLEMENTATION DEFINED IRS restore mechanism. See 4.11 *IRS power management* for more information.

$R_{PFQHM}$ When a level 2 IST becomes valid, the behavior is UNPREDICTABLE if all of the following are true for one or more level 2 IST entries:

- L2_ISTE.IRM is 1.
- L2_ISTE.IAFFID is not 0.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{VZCQM}$ To recover from UNPREDICTABLE behavior resulting from a level 2 IST containing entries with a non-zero 1ofN interrupt hint becoming valid, the IST is made invalid.

$I_{KJLYK}$ A level 2 IST containing non-zero 1ofN interrupt hints may become valid as part of an IMPLEMENTATION DEFINED IRS restore mechanism. See 4.11 *IRS power management* for more information.

### 4.7.3 INTID state and configuration

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{PTGRK}$ For each INTID in the configured range for an IST, there is a corresponding level 2 IST entry, if the IST is valid and one of the following is true:

- The IST uses a linear structure.
- The IST uses a 2-level structure and the level 2 IST is valid.

Otherwise, there is no level 2 IST entry corresponding to the INTID.

$R_{RFGXL}$ When a level 2 IST becomes valid, the configuration of each INTID is consistent with the values stored in the corresponding IST entry.

$R_{RFXND}$ When the IST is valid, the state and configuration of each INTID in the configured range for the IST are coherent across all IRSs.

$R_{NQXGQ}$ After a level 2 IST becomes valid, the values stored in the level 2 IST entries are UNKNOWN.

The values stored in the level 2 IST entries remain UNKNOWN after the IST becomes invalid.

$R_{BTYMB}$ If software performs a write to a level 2 IST entry when the IST is valid, the behavior is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in *4.2 IRS memory access rules*.

$R_{XLYLQ}$ To recover from UNPREDICTABLE behavior resulting from an incorrect write to a level 2 IST entry, the IST is made invalid.

### 4.7.4 IST metadata

$I_{RCFTT}$ IRS_IDR2.ISTMD reports whether an IRS stores metadata in level 2 IST entries.

$D_{JKSZH}$ If an IRS stores metadata in the level 2 IST entries, the bytes containing the metadata is called the *metadata area*.

$R_{WDNFJ}$ The value of IRS_IDR2.ISTMD is the same for all IRSs for each Interrupt Domain in the system.

$I_{HPJWB}$ The size of a level 2 IST entry is 4 bytes, 8 bytes, or 16 bytes.

If the size of a level 2 IST entry is more than 4 bytes, the higher address bytes are RES0.

The layout of the 4 lowest address bytes is defined by the architecture in L2_ISTE, Level 2 interrupt state table entry.

$I_{CMFVK}$ The metadata area is stored in the highest address bytes of the IST entry.

$I_{QRFDQ}$ IRS_IDR2.ISTMD_SZ describes the minimum number of INTID.ID bits which requires a level 2 IST entry size of 16 bytes to store metadata.

$R_{THDPW}$ The value of IRS_IDR2.ISTMD_SZ is the same for all IRSs for all Interrupt Domains in the system.

$I_{FFWQC}$ The minimum size of the level 2 IST entries used for the physical IST depends on the whether the IRS stores metadata in the level 2 IST entries and on the number of LPIs stored in the IST.

IRS_IDR2.ISTMD reports whether the IRS stores metadata in the level 2 IST entries.

The minimum size of the level 2 IST entries is 4 bytes, 8 bytes, or 16 bytes as follows:

- If the IRS does not store metadata in the IST, the minimum size of the level 2 IST entries is 4 bytes.
- If the IRS stores metadata in the IST and the number of INTID.ID bits is less than the value in IRS_IDR2.ISTMD_SZ, the minimum size of the level 2 IST entries is 8 bytes.
- If the IRS stores metadata in the IST and the number of INTID.ID bits is greater than or equal to IRS_IDR2.ISTMD_SZ, the minimum size of the level 2 IST entries is 16 bytes.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 4.7.5 Example IST structures

I_SSCKN The size of a level 1 IST entry is 8 bytes and the size of a level 2 IST entry may scale with the number of INTIDs stored in the IST.

Table 4.2 shows some example IST structures for an implementation that does not require storing metadata in the IST and therefore the size of the level 2 IST entry does not scale with the configured number of INTIDs.

Table 4.3 shows some example IST structures for an implementation that requires storing metadata in the IST. The examples assume that the size of a level 2 IST entry is 8 bytes up to and including 16 bits of LPI ID, and 16 bytes for more than 16 bits of LPI ID.

S_BRMDG The L1 size and L2 size columns in Table 4.2 and Table 4.3 indicate the required maximum physically contiguous allocation by software for the configuration.

**Table 4.2: Example IST structures without metadata storage**

| STRUCTURE | LPI_ID_BITS | L2_ISTE size | L2SZ | L1 size | L2 size | Maximum number of LPIs |
|---|---|---|---|---|---|---|
| Linear | 10 | 4 bytes | - | - | 4 KB | 1024 |
| Linear | 14 | 4 bytes | - | - | 64 KB | 16,384 |
| Linear | 16 | 4 bytes | - | - | 256 KB | 65,536 |
| 2-level | 19 | 4 bytes | 0b00, 9 bits | 4 KB | 4 KB | 524,288 |
| 2-level | 24 | 4 bytes | 0b00, 10 bits | 128 KB | 4 KB | 16,777,216 |
| 2-level | 16 | 4 bytes | 0b01, 12 bits | 128 B | 16 KB | 65,536 |
| 2-level | 24 | 4 bytes | 0b01, 12 bits | 32 KB | 16 KB | 16,777,216 |
| 2-level | 16 | 4 bytes | 0b10, 14 bits | 32 B | 64 KB | 65,536 |
| 2-level | 24 | 4 bytes | 0b10, 14 bits | 8 KB | 64 KB | 16,777,216 |

**Table 4.3: Example IST structures with metadata storage**

| STRUCTURE | LPI_ID_BITS | L2_ISTE size | L2SZ | L1 size | L2 size | Maximum number of LPIs |
|---|---|---|---|---|---|---|
| Linear | 9 | 8 bytes | - | - | 4 KB | 512 |
| Linear | 13 | 8 bytes | - | - | 64 KB | 8,192 |
| Linear | 16 | 8 bytes | - | - | 512 KB | 65,536 |
| 2-level | 16 | 8 bytes | 0b00, 9 bits | 1 KB | 4 KB | 65,536 |
| 2-level | 17 | 16 bytes | 0b00, 8 bits | 4 KB | 4 KB | 131,072 |
| 2-level | 24 | 16 bytes | 0b00, 8 bits | 512 KB | 4 KB | 16,777,216 |
| 2-level | 16 | 8 bytes | 0b01, 11 bits | 256 B | 16 KB | 65,536 |
| 2-level | 24 | 16 bytes | 0b01, 10 bits | 128 KB | 16 KB | 16,777,216 |
| 2-level | 16 | 8 bytes | 0b10, 13 bits | 64 B | 64 KB | 65,536 |
| 2-level | 24 | 16 bytes | 0b10, 12 bits | 32 KB | 64 KB | 16,777,216 |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{LWNFZ}$ For a 2-level IST, an implementation may only implement support for a subset of the possible L2SZ values. The supported L2SZ values are reported by IRS_IDR2.IST_L2SZ.

$R_{FVVBH}$ The value of IRS_IDR2.IST_L2SZ is the same for all IRSs for all Interrupt Domains in the system.

$I_{MCFNW}$ IRS_IDR2.{MIN_LPI_ID_BITS,ID_BITS} report the range of valid values for IRS_IST_CFGR.LPI_ID_BITS.

See also:

- 11.2 *IRS Data Structures*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.8 Physical interrupts

$D_{RVLFH}$ *Physical interrupts* are interrupts that belong to a Physical Interrupt Domain.

$R_{ZQGCF}$ A physical interrupt is only permitted to be selected as the candidate HPPI for the corresponding Physical Interrupt Domain.

$I_{NYGDM}$ For a description of how the IRS supports virtualization, see 4.10 *Virtual interrupts*.

### 4.8.1 Physical LPIs

$R_{VWVXB}$ The physical IST contains the configuration and state for the physical LPIs in an IRS Domain.

$I_{BHYYV}$ When support for physical LPIs is not implemented, there is no physical IST. IRS_IDR2.LPI reports whether support for physical LPIs is implemented. See 4.6 *Interrupt configuration* for more information.

$I_{XZPJV}$ The number of LPIs in an IRS Domain is configured in IRS_IST_CFGR.LPI_ID_BITS.

Software allocates the physical IST for an IRS Domain by writing a valid address to IRS_IST_BASER.ADDR and writing 1 to IRS_IST_BASER.VALID to make the IST valid.

$R_{QSXDB}$ The physical IST is indexed using an INTID.ID value in the physical LPI range, from 0 to (2 ^ IRS_IST_CFGR.LPI_ID_BITS) - 1.

$R_{TXDPZ}$ When an IRS accesses the physical IST, the IRS does not access any memory location derived from an INTID which is outside the configured LPI range.

This includes situations where the INTID is programmed into parameters of a GIC System instruction executed on a PE or the INTID is stored in the IST metadata areas.

$R_{LFZVT}$ An INTID specifying an LPI in the physical Interrupt Domain is reachable, if all of the following are true:

- Support for physical LPIs is implemented.
- The physical IST is valid.
- There is a level 2 IST entry corresponding to the INTID.
- The INTID is in the configured physical LPI range for the IRS Domain.

Otherwise, the INTID is unreachable.

$I_{MZRSK}$ In the physical LPI IST context, the base address of the physical IST for an IRS Domain is stored in IRS_IST_BASER.ADDR.

$I_{XCVJT}$ In the physical LPI IST context, IRS_IST_BASER.VALID determines whether the physical IST is valid.

$I_{KYYPW}$ A write to IRS_IST_BASER completes when IRS_IST_STATUSR.IDLE is 1.

$S_{KMPNW}$ The architecture relies on making a physical IST valid or invalid to support the following software sequences:

**Initial configuration after system reset** Software is expected to make the IST valid as part of the boot process.
**Soft reboot** Software can reclaim the memory used for the IST by making the IST invalid.

Making an IST valid or invalid is not intended to be used in power management sequences. See 4.11 *IRS power management* for more information about power management of an IRS.

$I_{ZRRXD}$ In a multi-IRS system, the physical IST is shared across all IRSs for an IRS Domain.

$D_{MRGCV}$ The physical IST *becomes valid* when a write that updates IRS_IST_BASER.VALID from 0 to 1 completes.

The physical IST *becomes invalid* when a write that updates IRS_IST_BASER.VALID from 1 to 0 completes.

See 4.7 *The interrupt state table (IST)* for more information about an IST becoming valid and invalid.

$R_{NSNNN}$ In a multi-IRS system, when a write to IRS_IST_BASER.VALID completes, a subsequent read of any of the following registers on any IRS in the IRS Domain returns the same value.

- IRS_IST_BASER.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- IRS_IST_CFGR.

$R_{KLBCH}$ If a write occurs to IRS_IST_BASER.VALID when IRS_IST_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE which configuration and base address is used for the IST.

The CONSTRAINED UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the IRS Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{SSTWT}$ When the physical IST becomes invalid and an LPI for the IRS Domain is selected as the candidate HPPI for a PE for the Interrupt Domain, one of the following is true:

- The candidate HPPI is acknowledged by the PE before the IST becomes invalid.
- The candidate HPPI is not acknowledged and is no longer selected as the candidate HPPI for the PE.

$R_{LJXGD}$ When the physical IST becomes invalid, if there are Accepted events for ph[illegible]LPIs that are not yet processed, all of the following are true:

- The events are dropped and have no effect on any interrupt.
- If software error reporting is supported, the IRS is permitte[illegible]to rep[illegible]an error.

$I_{KQGDP}$ When the physical IST is valid, all IRSs are permitted t[illegible]ccess the phys[illegible]l [illegible] for any reason, including speculative reads.

When the physical IST is invalid, the IRSs do not access the p[illegible]ical IST.

See 4.7 *The interrupt state table (IST)* for mor[illegible]nforma[illegible]on.

$I_{JVVMS}$ The configuration of the physical IST is con[illegible]lled usi[illegible]the following register fields:

- IRS_IST_CFGR.STRUCTURE.
- IRS_IST_CFGR.LPI_ID_BIT[illegible]
- IRS_IST_CFGR.L2SZ.
- IRS_IST_CFGR.ISTSZ

$I_{MBKRT}$ When IRS_IDR2.IST_levels is 0, IR[illegible]IST_CFGR.STRUCTURE is RES0.

$I_{CLPKT}$ IRS_IDR2.{MIN_L[illegible]_ID_BITS,ID_BIT[illegible]report the range of valid values for IRS_IST_CFGR.LPI_ID_BITS.

$I_{TQCYG}$ In the physical I[illegible]IST co[illegible]xt, when the physical IST is valid and uses a 2-level structure, a physical level 2 IST is made valid by w[illegible]in[illegible]o IRS_M[illegible]P_L2_ISTR.

The effe[illegible]of [illegible]write [illegible]e com[illegible]ete when IRS_IST_STATUSR.IDLE is 1.

$R_{CMYTZ}$ If a[illegible]ite occur[illegible]to IRS_M[illegible]P_L2_ISTR when IRS_IST_STATUSR.IDLE is 0 on any other IRS for the same IRS Dom[illegible]it i[illegible]ED UNPREDICTABLE whether any effects of any of the writes complete successfully.

See also:

- 4.7 *The [illegible]errupt state table (IST)*
- 11.2 *IRS Data Structures*

### 4.8.2 Physical SPIs

$G_{BJJKB}$ SPIs are designed to enable implementations to meet the following requirements:

- SPIs can be used to connect interrupt signals directly to an IRS, without recourse to an ITS.
- SPIs can generate interrupts to PEs without requiring memory being allocated to the IRS.
- SPIs can reliably generate interrupts to PEs in the presence of memory system errors.
- SPIs can be implemented to provide predictable low latency guarantees for interrupt delivery.
- SPIs can be assigned to VMs to support system partitioning using virtualization techniques.

$R_{TLLZS}$ The storage location for physical SPIs is IMPLEMENTATION DEFINED.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{ZFRNY}$ The number of SPIs across all IRSs in the system across all Interrupt Domains is reported by IRS_IDR5.SPI_RANGE.

$R_{KXMTR}$ The value reported by IRS_IDR5.SPI_RANGE is the same across all Interrupt Domains across all IRSs in a system.

$R_{MHRZT}$ An INTID specifying an SPI in a Physical Interrupt Domain is reachable, when all of the following are true:

- The SPI is implemented.
- The SPI is assigned to the Interrupt Domain.
- The SPI is not assigned to a VM.
- The INTID.ID is less than or equal to IRS_IDR5.SPI_RANGE - 1.

Otherwise, the INTID is unreachable.

$I_{BVXHP}$ See 4.10.2.1 *Assigning physical SPIs to VMs* for more information about assigning an SPI to a VM.

$I_{HZSKP}$ The configuration and state of a reachable SPI can be accessed via the GIC instructions on the PE.

$D_{KZSTV}$ An SPI is *managed* by a single IRS.

This means that an SPI can be assigned to an Interrupt Domain and to a VM only in the IRS where the SPI is managed.

$R_{VDLHR}$ Each SPI is identified by a unique INTID.

$I_{XYNMZ}$ The range of SPIs managed by an IRS is reported in IRS_IDR6.SPI_IRS_RANGE

If IRS_IDR6.SPI_IRS_RANGE is 0, the IRS does not manage any SPIs, and IRS_IDR7.SPI_BASE is RES0.

The minimum INTID.ID of the SPIs managed by a individual IRS is reported by IRS_IDR7.SPI_BASE.

The range of SPI INTIDs managed by the IRS can be calculated as IRS_IDR7.SPI_BASE + IRS_IDR6.SPI_IRS_RANGE - 1.

$I_{YMKMH}$ If IRS_IDR5.SPI_RANGE is 0, no SPIs are implemented on any IRS, and all of the following are true:

- IRS_IDR6.SPI_IRS_RANGE is RES0.
- IRS_IDR7.SPI_BASE is RES0.

$R_{GCVXZ}$ Not all SPIs in the range reported by IRS_IDR7.SPI_BASE and IRS_IDR6.SPI_IRS_RANGE are required to be implemented.

$R_{CSQHP}$ The number reported by IRS_IDR6.SPI_IRS_RANGE is the same across all Interrupt Domains.

The number reported by IRS_IDR7.SPI_BASE is the same across all Interrupt Domains.

$R_{ZYGST}$ For each IRS, all of the following are true for the range reported by IRS_IDR7.SPI_BASE and IRS_IDR6.SPI_IRS_RANGE:

- The range does not overlap with the range reported by other IRSs in the system.
- The minimum and maximum SPI INTID included in the range are less than IRS_IDR5.SPI_RANGE.

$I_{QNBQC}$ It is permitted to implement SPIs that are not connected to any interrupt input signal. Such SPIs can only be made Pending as a result of receiving a SET interrupt event from a PE.

This may be useful to support IPIs without requiring the configuration and state for IPIs to be stored in memory.

See also 2.5 *Inter-Processor Interrupts*.

$R_{PDMTJ}$ When an SPI is assigned to an Interrupt Domain, all of the following are true:

- The SPI is only considered for a candidate HPPI in the Interrupt Domain that the SPI is assigned to.
- The SPI is only reachable in the Interrupt Domain that the SPI is assigned to.

$I_{YTLRC}$ This means that an SPI is unreachable from any Interrupt Domain other than the SPI is assigned to.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{HWBBY}$ The *SPI IRS configuration* for each SPI consists of the following configurations, which are separate from the SPI INTID configuration and state:

- The assignment of an SPI to an Interrupt Domain.
- The assignment of an SPI to a VM.
- The Trigger mode of the SPI.

$R_{ZYJHT}$ The SPI IRS configuration for an SPI can be accessed from the following IRS Domains:

- The IRS Domain correponding to the Interrupt Domain that the SPI is assigned to.
- The EL3 IRS Domain.

$I_{SZZNP}$ Software accesses the SPI IRS configuration state using one of the following registers after selecting an implemented SPI managed by the IRS:

- IRS_SPI_CFGR.
- IRS_SPI_DOMAINR.
- IRS_SPI_VMR.

Software selects an SPI by writing the SPI INTID to IRS_SPI_SELR.

IRS_SPI_STATUSR.IDLE reports whether the effects of a write to any of the SPI registers is completed.

IRS_SPI_STATUSR.V reports whether the value written to IRS_SPI_SELR selects an implemented SPI that is managed by the IRS and where the configuration can be accessed in the IRS Domain.

IRS_SPI_STATUSR.V is updated on a write to any of the following registers:

- IRS_SPI_CFGR.
- IRS_SPI_SELR.
- IRS_SPI_VMR.

If an SPI is assigned to a new Interrupt Domain after being selected by IRS_SPI_SELR, then a subsequent write to any of the registers above in the new Interrupt Domain causes IRS_SPI_STATUSR.V to become 0.

$I_{RSFPY}$ When an SPI is selected using IRS_SPI_SELR and IRS_SPI_STATUSR.{V,IDLE} is {1,1}, IRS_SPI_CFGR.TM provides access to the Trigger mode of the SPI.

The Trigger mode of an SPI determines when the IRS generates events for an SPI and defines the type of the events.

Following a write that updates IRS_SPI_CFGR.TM to 1, the effects of the write are complete when IRS_SPI_STATUSR.{V,IDLE} is {1,1}.

See *6 Interrupt configuration and state* for more information about the Handling mode.

$R_{QBXXV}$ When the Trigger mode of an SPI is edge-triggered, the IRS generates a SET_EDGE event when the interrupt signal connected to an SPI is asserted.

When the Trigger mode of an SPI is level-sensitive, the IRS generates a SET_LEVEL event when the interrupt signal connected to an SPI is asserted and a CLEAR event when the signal is de-asserted.

The IRS does not generate a CLEAR event when the Trigger mode is edge-triggered and the signal is de-asserted.

$I_{BGJKL}$ For each SPI, it is IMPLEMENTATION DEFINED whether the Trigger mode can be programmed by software.

If the Trigger mode cannot be programmed by software, access to the corresponding field is RO.

$S_{DCPVJ}$ Software can detect that the Trigger mode of an SPI cannot be programmed by attempting to update the Trigger mode and reading it back. If the Trigger mode cannot be programmed, the Trigger mode is not updated.

$R_{KBPXL}$ When the Trigger mode of an SPI is updated from level-sensitive to edge-triggered and the interrupt signal is asserted, the IRS generates a CLEAR event for the SPI.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

For an SPI connected to an interrupt signal, when the Trigger mode of the SPI is updated from edge-triggered to level-sensitive, the IRS generates a SET_LEVEL event if the interrupt signal is asserted, and a CLEAR event if the signal is de-asserted.

For an SPI not connected to an interrupt signal, when the Trigger mode of the SPI is updated from edge-triggered to level-sensitive, the IRS generates a CLEAR event for the SPI.

When an update of the Trigger mode of an SPI generates an Interrupt Write Effect $E_1$ to the SPI Interrupt Location, all of the following are true:

- There is a Register Write Effect $E_2$ to IRS_SPI_STATUSR which updates the IDLE field from 1 to 0.
- There is a Register Write Effect $E_3$ to IRS_SPI_STATUSR which updates the IDLE field from 0 to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_2$ to the Interrupt Write Effect $E_1$.
- There is an Intrinsic order dependency from the Interrupt Write Effects $E_1$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_1$ is Ordered-before $E_4$.

$R_{LKBBQ}$ When the Trigger mode of an SPI is level-sensitive and connected to an interrupt input signal, the IRS is permitted to IGNORE SetPending commands received from PEs specifying that SPI

$D_{XKYXL}$ A write to IRS_SPI_RESAMPLER causes the IRS to *resample* the signal connected to an SPI, if all of the following are true:

- The SPI is managed on the IRS where the write occurs.
- The SPI IRS configuration can be accessed in the IRS Domain where the write occurs.

Otherwise, the write has no effect beyond reporting a software error using EC 0x2B if supported.

$R_{DMTFM}$ When the signal of an SPI is resampled, all of the following are true:

- If the SPI Trigger mode is level-sensitive, all of the following are true:
  - If the signal is asserted at the time of the resample, the IRS generates a SET_LEVEL event.
  - If the signal is de-asserted at the time of the resample, the IRS generates a CLEAR event.
- If the SPI Trigger mode is edge-triggered, all of the following are true:
  - If the signal is asserted at the time of the resample, the IRS generates a SET_EDGE event.
  - If the signal is de-asserted at the time of the resample, the IRS generates no events.

$R_{VYMHB}$ When a resample of an SPI generates an Interrupt Write Effects $E_1$ to the SPI Interrupt Location, all of the following are true:

- There is a Register Write Effect $E_2$ to IRS_SPI_STATUSR which updates the IDLE field from 1 to 0.
- There is a Register Write Effect $E_3$ to IRS_SPI_STATUSR which updates the IDLE field from 0 to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_2$ to the Interrupt Write Effects $E_1$.
- There is an Intrinsic order dependency from the Interrupt Write Effects $E_1$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_1$ is Ordered-before $E_4$.

$S_{SNRXG}$ Software may need to resample the state of an SPI to ensure that the Pending state of the SPI INTID corresponds to the state of the signal connected to the SPI. For example, if software has updated the Pending state or Handling mode of the SPI using GIC system instructions, it can use the resample mechanism to recover from this scenario.

Software may also need to resample an SPI if configuring the SPI for a source with level-sensitive semantics as Edge-triggered. In this case, software can resample the SPI after handling each event to ensure that interrupt events are not lost if the signal remained asserted while handling the interrupt.

$D_{YGLYC}$ The *reset state of the SPI IRS configuration* is the following:

- If the SPI Trigger mode can be programmed by software, it is reset to edge-triggered.
- The SPI is not assigned to a VM.
- The SPI is assigned to the most privileged IRS Domain.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

DTVVRZ The *reset state of an SPI INTID* is the following:

- The interrupt is Disabled.
- The interrupt is Inactive.
- The interrupt is Idle.
- If 1ofN interrupt routing is not supported, the SPI Routing mode is Targeted.
- All other interrupt configuration and state are UNKNOWN.

RQRHKF When an IRS is reset, all of the following are true:

- All SPI INTIDs are reset to their reset state.
- The IRS configuration for each SPI is reset to its reset state.

SHZPVX To configure the assignment of an SPI to a VM, software performs the following sequence:

1. Software selects the SPI by writing to IRS_SPI_SELR.
2. Software polls IRS_SPI_STATUS.IDLE until it reads as 1 and ensures that IRS_SPI_STATUS.V is 1.
3. Software accesses the VM assignment of the SPI by accessing IRS_SPI_VMR.
4. Following a write to IRS_SPI_VMR, software polls IRS_SPI_STATUS.IDLE until it reads as 1, and ensures that IRS_SPI_STATUS.V is 1, to ensure the effects of the write are complete.

SGQDWY To configure the assignment of an Interrupt Domain, software performs the following sequence using the EL3 Interrupt Domain IRS configuration frame:

1. Software selects the SPI by writing to IRS_SPI_SELR.
2. Software polls IRS_SPI_STATUS.IDLE until it reads as 1 and ensures that IRS_SPI_STATUS.V is 1.
3. Software accesses the Interrupt Domain assignment of the SPI by accessing IRS_SPI_DOMAINR.
4. Following a write to IRS_SPI_DOMAINR, software polls IRS_SPI_STATUS.IDLE until it reads as 1.

To check whether an SPI is statically assigned to an Interrupt Domain or it can be dynamically assigned to an Interrupt Domain, software can attempt to update the assignment of the SPI to an Interrupt Domain and read back the assigned domain from IRS_SPI_DOMAINR.DOMAIN when IRS_SPI_STATUSR.IDLE is 1.

IDMFCZ An SPI may be statically assigned to an Interrupt Domain.

IBCQRT Arm recommends only statically assigning SPIs to an Interrupt Domain when it is known that the interrupt can only be handled in that Domain. For example, RAS interrupts may be statically assigned to the EL3 Domain. Statically assigning interrupts to a Domain may restrict how that interrupt can be used. For example, static assignment may prevent the interrupt from being directly assigned to Realms.

RJJHJX When an SPI is assigned to a new Interrupt Domain as a result of selecting the SPI by writing to IRS_SPI_SELR and writing to IRS_SPI_DOMAINR.DOMAIN, the effects of the write are complete when IRS_SPI_STATUSR.IDLE is 1.

When the effects are complete, all of the following are true:

- If the SPI was selected as the candidate HPPI for a PE in the old Interrupt Domain, one of the following is true:
  - The SPI is acknowledged in the old Interrupt Domain before the SPI is assigned to the new Interrupt Domain.
  - The SPI is not acknowledged in the old Interrupt Domain and no longer selected as the candidate HPPI in the old Interrupt Domain.
- If the SPI was assigned to a VM in the old Interrupt Domain, it is CONSTRAINED UNPREDICTABLE whether the configuration and state of the virtual SPI INTID is UNKNOWN or reset to its reset state.
- The SPI is not assigned to a VM in the new Interrupt Domain.
- The SPI INTID is reset to its reset state.

See also:

- 3.1 *Interrupt Domains*
- 4.6 *Interrupt configuration and state*
- 4.7 *The interrupt state table (IST)*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- 4.8.4 *Physical interrupt signaling*

### 4.8.3 Physical interrupt routing

$R_{LFYMS}$ The Routing mode of a physical interrupt determines how the target PE is selected for the interrupt in the following ways:

- If the Routing mode is Targeted, the target PE is the PE specified by the interrupt Affinity.
- If the Routing mode is 1ofN, the target PE is determined dynamically.

$D_{VBQJG}$ When an interrupt Routing mode is Targeted and the interrupt Affinity specifies a PE, the interrupt is said to be *targeted to* that PE.

$R_{HGPQH}$ IRS support for 1ofN interrupt routing is optional.

$D_{VPZNM}$ An interrupt whose Routing mode is *1ofN* is referred to as a *1ofN interru*

$I_{GHKQS}$ IRS_IDR0.ONE_N reports whether 1ofN interrupt routing is suppor

$R_{FGVHX}$ The value of IRS_IDR0.ONE_N is the same for all IRSs for all I terrup Domains in e system.

$R_{ZMTLR}$ If 1ofN interrupt routing is not supported, all of the followi are true:

- If a GIC System instruction generates an Interrup Effect at updates the Routing mode of a physical interrupt, the Routing mode is set to Targeted regardles he value ovided in <Xt>.
- L2_ISTE.IRM is treated as 0 when the phys al beco s vali

$R_{GJNGF}$ Each physical 1ofN interrupt can be acknow ged by a most one

$R_{STMZL}$ The storage location for the Affinity of physica 1of interrupts ay be used for an IMPLEMENTATION DEFINED purpose to handle routing of physical fN inte pts, with the restriction that an Affinity value of 0 has no IMPLEMENTATION DEFINED mea g.

$R_{QZWMR}$ An IRS is permitted to update Affinity of ph ical 1ofN interrupts.

$R_{DJNNR}$ A PE may only be selected as the ta et for a 1ofN interrupt, if 1ofN PE selection is enabled for the PE for the Interrupt Domain

$R_{CDXNW}$ An IRS Domain s cts a tar t PE for a reachable 1ofN interrupt in finite time, when all of the following are true:

- The interrup Pe ing.
- The i upt is ctive.
- Th inter t is En led.
- t least o IRS is en ied.
- t leas ected to an enabled IRS has 1ofN PE selection enabled.

$R_{TMBZH}$ An IRS D main may select a new target PE for a physical 1ofN interrupt at any time.

$I_{DBQSC}$ Selecting a ta t PE does not guarantee that the interrupt is selected as the candidate HPPI for the target PE. For example, the selected target PE may be masking the interrupt or another interrupt may be the candidate HPPI for the selected target PE.

See 4.8.4 *Physical interrupt signaling* for more information about selecting the physical candidate HPPI.

$R_{FYRLW}$ The mechanism that an IRS uses to select a target PE among several possible target PEs is IMPLEMENTATION DEFINED.

$I_{LZSJY}$ The architecture does not require that higher priority 1ofN interrupt are delivered before lower priority 1ofN interrupts.

For example, the following situation may occur:

- The IRS selects PE 0 for a high priority 1ofN interrupt (INTID A).
- The IRS selects PE 1 for a low priority 1ofN interrupt (INTID B).
- There are several interrupts with higher priority than INTID A that are Targeted to PE 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

In this situation, PE 1 may handle INTID B before PE 0 handles INTID A.

$I_{XRMBH}$ Whether a PE has 1ofN PE selection enabled or disabled is configured separately for each IRS Domain.

This configuration setting is accessed using the following sequence:

1. Select the PE using IRS_PE_SELR on the IRS where the PE is connected.
2. Poll IRS_PE_STATUSR.IDLE until it reads as 1.
3. Access IRS_PE_CR0.DPS.
4. Poll IRS_PE_STATUSR.IDLE until it reads as 1.

$R_{LQMBW}$ When 1ofN selection is disabled for PE for an Interrupt Domain, the corresponding PE is not selected as the target for 1ofN interrupts belonging to that Interrupt Domain.

$I_{SHQTF}$ When IRS_PE_CR0.DPS is updated for a PE, and a physical 1ofN interrupt is selected as the physical candidate HPPI for the PE, one of the following is true:

- The PE acknowledges the interrupt before IRS_PE_STATUSR.IDLE is 1.
- The PE does not acknowledge the interrupt.

See 4.8.4 *Physical interrupt signaling* for more information.

### 4.8.4 Physical interrupt signaling

$R_{LXYDJ}$ The Pending state of a physical LPI identified by an INTID is updated when all of the following are true:

- The physical LPI as identified by the INTID is reachable.
- Any of the following occur:
  - An ITS generates an interrupt event that specifies the INTID.
  - A write to the IRS_SETLPI register specifies the INTID.
  - An IRS processes an Interrupt Effect generated by a GIC System instruction that updates the LPI Pending state.

Otherwise, the physical LPI Pending state is not updated.

$I_{DZSFL}$ The Interrupt Domain of an LPI that is being signaled is determined by how the interrupt is signaled as follows:

- The Interrupt Domain is specified as part of the interrupt event received from the ITS.
- The Interrupt Domain is reported in IRS_IDR0.INT_DOM for the IRS where a write to IRS_SETLPI occurs.
- The Interrupt Domain that the GIC System instruction operates in.

$R_{XYLLD}$ The Pending state of a physical SPI identified by an INTID is updated when all of the following are true:

- The physical SPI as identified by the INTID is reachable.
- Any of the following occur:
  - An interrupt connected to an IRS as an SPI is asserted or de-asserted and generates an interrupt event for the physical SPI.
  - An IRS processes an Interrupt Effect generated by a GIC System instruction that updates the SPI Pending state.

Otherwise, the physical SPI Pending state is not updated.

$I_{SKNHH}$ The Interrupt Domain of an SPI that is being signaled is determined by the Interrupt Domain that the SPI is assigned to.

See 4.8.2 *Physical SPIs* and 4.2 *Signaling interrupts* for more information.

$I_{SZZVX}$ If an SPI that is assigned to a VM is signaled, the SPI is a virtual SPI and the rules for when its Pending state is updated are different. See 4.10 *Virtual interrupts* and 4.10.4 *Virtual interrupt signaling* for more information.

$D_{NFYQX}$ The IRS Domain selects a physical candidate HPPI for a PE for an Interrupt Domain if, and only if, at least one physical interrupt meets the physical *candidate HPPI conditions*:

- The interrupt is Enabled.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The interrupt is Pending.
- The interrupt is Inactive.
- One of the following is true:
  - The interrupt Routing mode is Targeted and the interrupt Affinity specifies the PE.
  - The interrupt Routing mode is 1ofN and the PE is selected as the target PE for the interrupt.

$R_{WSBLV}$ For a PE and an Interrupt Domain, all of the following apply to the physical candidate HPPI selection:

- If at least one interrupt satisfies the physical candidate HPPI conditions, the IRS Domain selects one of these as the physical candidate HPPI in finite time.
- If no interrupts satisfy the physical candidate HPPI conditions, the IRS Domain does not select any interrupt as the physical candidate HPPI.

$R_{HGTSH}$ If more than one interrupt satisfies the physical candidate HPPI conditions for a PE for an Interrupt Domain, the IRS Domain selects the interrupt with the highest Priority as the physical candidate HPPI in finite time.

If there is more than one physical candidate HPPI with the same Priority, it is IMPLEMENTATION DEFINED which of those interrupts is selected as the physical candidate HPPI.

$I_{BGBFD}$ The architecture only requires that the highest Priority interrupt that satisfies the candidate HPPI conditions is selected in finite time, because the set of interrupts that satisfy the conditions changes over time as the state and configuration of interrupts are updated. An implementation may need to consider the Priority of all the interrupts that satisfy the conditions which may require processing time.

$R_{KFYBX}$ When at least one physical interrupt meets the *physical candidate HPPI conditions* for a PE for an Interrupt Domain, and the IRS is enabled for the Interrupt Domain, the IRS selects a physical candidate HPPI in finite time.

$I_{PZVZZ}$ See 4.11 *IRS power management* for information about the IRS behavior if it has selected a physical candidate HPPI for a PE and the PE is offline.

$I_{ZPJZL}$ An interrupt might be signaled at one IRS and be targeted to a PE connected to a different IRS. Whether that interrupt can be selected as the physical candidate HPPI for the PE depends only on the configuration of the IRS local to the PE. If the IRS in which the interrupt was originally signaled is subsequently disabled, that has no effect on whether the interrupt is selected as the physical candidate HPPI for the targeted PE.

$R_{XWWNP}$ When a physical interrupt that has been selected as the physical candidate HPPI for a PE for an Interrupt Domain is no longer selected as the physical candidate HPPI, the PE will either acknowledge the old physical candidate HPPI within finite time or the old physical candidate HPPI will not be acknowledged.

$R_{CHNVV}$ If the IRS updates the configuration of an interrupt that has been selected as the physical candidate HPPI for an Interrupt Domain for a PE, and the interrupt configuration data was communicated to the PE, the IRS will select the same or a different physical interrupt as the physical candidate HPPI.

The IRS then communicates the selected physical candidate HPPI to the PE, including any updated interrupt configuration data, in finite time.

For more information about ordering of such events, see 2.12 *Interrupt ordering model and synchronization requirements* and Chapter B1 *Interrupt ordering litmus tests*.

---

**Note**

The following sequence is an example of the IRS communicating updated interrupt configuration data because there was an update to the configuration data that was communicated to a PE:

1. An interrupt is selected as the physical candidate HPPI for a PE and its Priority is communicated to the PE.
2. The PE executes a GIC System instruction that updates the Priority of the interrupt.
3. The IRS selects the same interrupt as the physical candidate HPPI for the PE.
4. The IRS communicates that the interrupt is still the candidate HPPI with the updated Priority value to the PE.

---

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{TMRWF}$ Updates to interrupt state or configuration can lead to the IRS selecting a different candidate HPPI. For example, the current candidate HPPI may become Disabled, or a higher priority interrupt may be updated to meet the candidate HPPI conditions. In such scenarios, the IRS selects a new candidate HPPI in finite time. However, the PE may acknowledge the old candidate HPPI before the IRS communicates to the PE that a new candidate HPPI has been selected.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.9 Virtualization data structures

IPKMJQ Figure 4.3 shows an overview of the data structures used to support VMs in the IRS.

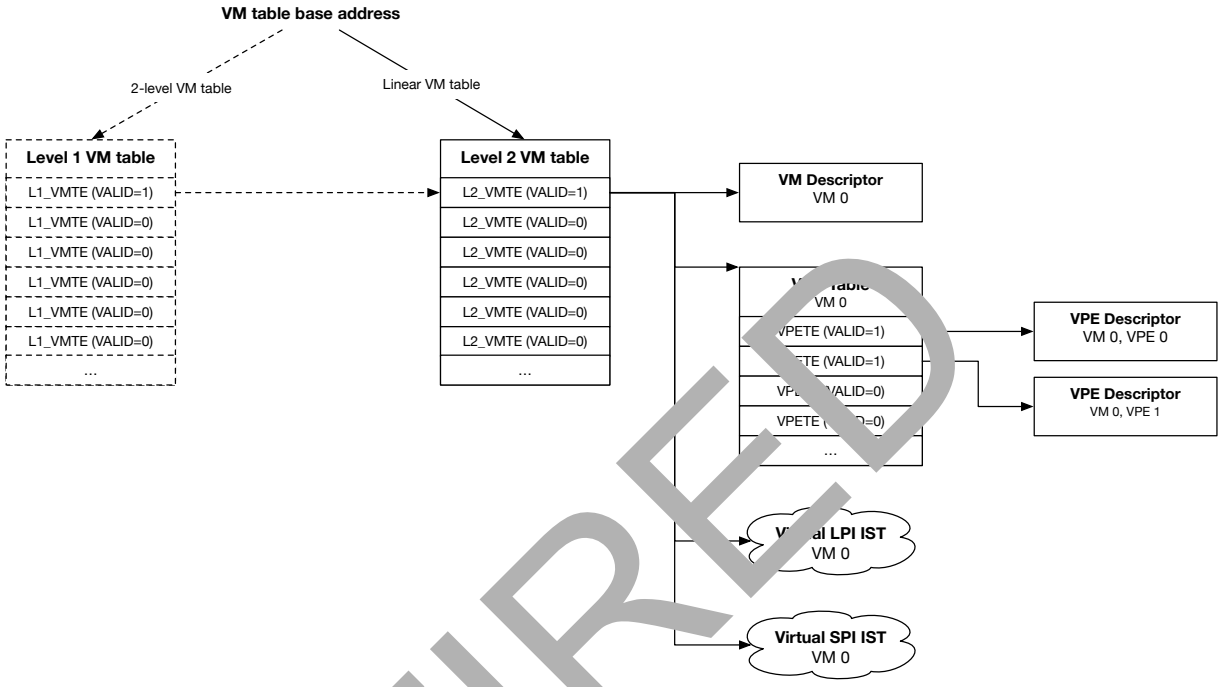



**Figure 4.3: IRS virtualization structures**

DCCSJV The GICv5 architecture defines the following *virtualization data structures* used by an IRS:

- The Virtual Machine table (VM table).
- The Virtual PE table (VPE table).

Each level 2 entry in a VM table describes the configuration of a VM and contains pointers to the following child data structures:

- A VM descriptor, if required.
- A VPE table.
- A virtual LPI IST.
- A virtual SPI IST.

Each entry in the VPE table describes the configuration of the VPE and contains pointers to the VPE descriptor for each VPE in the VM.

IBTVZZ A VM supports virtual LPIs, virtual SPIs, or both.

IYTSWN Virtual LPIs and virtual SPIs are stored in separate virtual ISTs.

SWXHNY Software is responsible for allocating the virtualization data structures from memory in the PAS associated with the IRS Domain where the VM table is used.

See also:

- 4.7 *The interrupt state table (IST)*
- 4.10 *Virtual interrupts*
- 11.2 *IRS Data Structures*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 4.9.1 The VM table

$I_{PVHPD}$ The VM table contains the configuration of the VMs defined in an IRS Domain.

For each VM, the VM table contains pointers to other data structures associated with that VM.

In a multi-IRS system, the VM table is shared between all IRSs for the IRS Domain.

$D_{PZXMY}$ The VM table uses either a linear or 2-level structure.

For a linear VM table, all of the following are true:

- The level 1 VM table is omitted.
- The VM table consists of a single level 2 VM table.

$I_{DSPSD}$ IRS_IDR3.VMT_LEVELS reports whether 2-level VM table support is implemented.

$R_{KQYKZ}$ The levels supported for the VM table are the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

The levels supported for the VM table are the same across all IRSs.

$D_{NRLZZ}$ For a linear VM table, the base address of the VM table specifies the location of the level 2 VM table.

For a 2-level VM table, the base address of the VM table specifies the location of the level 1 VM table.

$D_{LSPBK}$ When the VM table uses a 2-level table structure, the size of each level 2 VM table is 4 KB.

$R_{MKPCF}$ The structure of the VM table is controlled using the following register fields:

- IRS_VMT_CFGR.STRUCTURE:

  Selects if the VM table uses a linear or 2-level structure.

- IRS_VMT_CFGR.VM_ID_BITS:

  Selects how many VMs the IRS Domain supports. This impacts the size of the level 2 VM table when using a linear structure and the size of the level 1 VM table when using a 2-level structure.

$I_{HCGJW}$ The VM table is indexed using a VM ID ranging from 0 through (2 ^ IRS_VMT_CFGR.VM_ID_BITS) - 1.

$R_{NXJZZ}$ When an IRS accesses the VM table for an IRS Domain, the IRS does not access any memory location derived from a VM ID which is outside the configured VM ID range for the IRS Domain.

This includes situations where the VM ID is programmed in the parameters to a GIC System instruction, the VM ID is received from an ITS, or the VM ID is stored in a physical SPI VM assignment register.

$I_{ZKGTX}$ The maximum VM ID range supported is reported by IRS_IDR3.VM_ID_BITS.

$R_{XYKND}$ The maximum number of VM ID bits are the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

The maximum number of VM ID bits is the same across all IRSs.

#### 4.9.1.1 The VM table base address and configuration registers

$I_{GMXKS}$ The base address of the VM table is stored in IRS_VMT_BASER.ADDR.

$I_{JLYPB}$ IRS_VMT_BASER.VALID determines whether the VM table is valid.

$I_{BNSSH}$ A write to IRS_VMT_BASER completes when IRS_VMT_STATUSR.IDLE is 1.

$I_{YJKRP}$ In a multi-IRS system, the VM table is shared across all IRSs for an IRS Domain.

$D_{VJFSP}$ The VM table becomes valid when a write that updates IRS_VMT_BASER.VALID from 0 to 1 completes.

The VM table becomes invalid when a write that updates IRS_VMT_BASER.VALID from 1 to 0 completes.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{RLYYJ}$ In a multi-IRS system, when a write to IRS_VMT_BASER.VALID completes, the values of the following registers on that IRS are returned on a subsequent read on any IRS in the IRS Domain:

- IRS_VMT_BASER.
- IRS_VMT_CFGR.

$R_{WXTKP}$ If a write occurs to IRS_VMT_BASER.VALID when IRS_VMT_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE which configuration and base address is used for the VM table.

The CONSTRAINED UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the IRS Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{QRSZL}$ When the VM table is valid, all IRSs are permitted to access the virtualization data structures for any reason, including speculative reads.

When the VM table is invalid, the IRSs do not access the virtualization data structure table.

See 4.9 *Virtualization data structures* for the definition of virtualization data structure.

### 4.9.1.2 Level 2 VM table management

$D_{YPLVV}$ A level 2 VM table is considered *valid*, if the VM table is valid and one of the following is true:

- The VM table uses a linear structure.
- The VM table uses a 2-level structure and the corresponding L1_VMTE.VALID is 1.

Otherwise, the level 2 VM table is considered *invalid*.

$D_{QGKXL}$ A level 2 VM table *becomes valid* in any of the following scenarios:

- The VM table uses a linear structure and the VM table becomes valid.
- The VM table uses a 2-level structure and any of the following occur:
  - A write to IRS_VMAP_L2_VMTR makes the level 2 VM table valid.
  - The VM table becomes valid and the corresponding L1_VMTE.VALID is 1.

$I_{CSRYH}$ When a 2-level VM table becomes valid, if L1_VMTE.VALID is 1 for more than one level 1 VM table entry, more than one level 2 VM table may become valid at the same time.

$R_{BXWBF}$ When the VM table is valid and uses a 2-level structure, if software writes to an invalid level 1 VM table entry and sets L1_VMTE.VALID to 1, the behavior of the IRS Domain is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{XBDXW}$ For a 2-level VM table, if a write updates a valid level 1 VM table entry, the IRS Domain behavior is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the IRS Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$I_{DPCRG}$ The architecture does not support making an individual level 2 VM table invalid when using a 2-level structure for the VM table.

$R_{RNSYR}$ To recover from UNPREDICTABLE behavior resulting from an incorrect write to a level 1 VM table entry, the VM table is made invalid.

$R_{TZWTX}$ When a write to IRS_VMAP_L2_VMTR makes the corresponding level 2 VM table valid, all of the following are true:

- The IRS generates the following Effects:
  - A Register Write Effect $E_1$ to IRS_VMT_STATUSR which sets the IDLE field to 0.
  - A Memory Write Effect $E_2$ to the Location corresponding to the level 1 VM table entry which updates L1_VMTE.VALID from 0 to 1.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

  - A Register Write Effect $E_3$ to IRS_VMT_STATUSR which sets the IDLE field to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_1$ to the Memory Write Effect $E_2$.
- There is an Intrinsic order dependency from the Memory Write Effect $E_2$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_2$ is Ordered-before $E_4$.

$I_{YTWVR}$ When the VM table is valid and uses a 2-level structure, a level 2 VM table is made valid by the following sequence:

1. Software writes the address of the new level 2 VM table to L1_VMTE.L2_ADDR, with L1_VMTE.VALID remaining to be 0.
2. Software writes to IRS_VMAP_L2_VMTR.
3. The IRS performs a write to the level 1 VM table entry that updates L1_VMTE.VALID from 0 to 1.
4. The IRS reports that the effects of the write to IRS_VMAP_L2_VMTR are complete.

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

$R_{KZZPQ}$ When the VM table is valid and uses a 2-level structure, if software writes to the level 1 VM table entry when IRS_VMT_STATUSR.IDLE is 0, it is CONSTRAINED UNPREDICTABLE whether the IRS writes back the entry using the updated or old level 1 entry value.

The CONSTRAINED UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{KGTMJ}$ If a write occurs to IRS_VMAP_L2_VMTR when IRS_VMT_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE whether the effects of any of the writes complete successfully.

$D_{CFNYD}$ A level 2 VM table *becomes invalid* when the VM table becomes invalid.

#### 4.9.1.3 VM management

$D_{VBQTG}$ A VM and the corresponding level 2 VM table entry are considered *valid* when all of the following are true:

- The level 2 VM table is valid.
- The corresponding L2_VMTE.VALID is 1.

Otherwise the VM and the corresponding level 2 VM table entry are considered *invalid*.

$R_{YBMTV}$ If a VM ID is outside the configured VM ID range for the IRS Domain, the VM ID is treated as invalid.

$I_{JQFYM}$ When a VM is valid, all of the following are true:

- Virtual interrupts in the VM can be signaled.
- VMs can become resident.
- VPEs can access the configuration and state of reachable virtual interrupts.

When a VM is invalid, none of the above are true.

$I_{JFWCL}$ When the VM table is invalid, all of the following are true:

- All VM IDs specify invalid VMs.
- All VPEs are invalid.
- The IRS does not select any virtual candidate HPPIs to the PEs connected to the IRS.
- When a PE executes a GIC System instruction that operates in the Virtual Interrupt Domain, all Interrupt Effects generated by that instruction, that are Ordered-after the VM table becomes invalid, are IGNORED.

See Chapter 2 *PE architecture* for more information about the relationship between a resident VPE and the Virtual Interrupt Domain.

$D_{QXMZT}$ A VM *becomes valid* in any of the following scenarios:

- A write to IRS_VMAP_VMR makes the VM valid.
- The level 2 VM table becomes valid and the corresponding L2_VMTE.VALID is 1.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{VQKZB}$ When a level 2 VM table is valid, if software writes to an invalid level 2 VM table entry and sets L2_VMTE.VALID to 1, the behavior of the IRS Domain is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{HPDGY}$ To recover from UNPREDICTABLE behavior resulting from a write to an invalid level 2 VM table entry that sets L2_VMTE.VALID to 1, the VM table is made invalid.

$R_{KBKBF}$ When a write to IRS_VMAP_VMR makes the corresponding VM valid, all of the following are true:

- The IRS generates the following Effects:
  - A Register Write Effect $E_1$ to IRS_VMT_STATUSR which sets the IDLE field to 0.
  - A Memory Write Effect $E_2$ to the Location corresponding to the level 1 VM table entry which updates L2_VMTE.VALID from 0 to 1.
  - A Register Write Effect $E_3$ to IRS_VMT_STATUSR which sets the IDLE field to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_1$ to the Memory Write Effect $E_2$.
- There is an Intrinsic order dependency from the Memory Write Effect $E_2$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_2$ is Ordered-before $E_4$.

$I_{LTQMV}$ When a level 2 VM table is valid, a VM is made valid by the following sequence:

1. Software initializes the corresponding level 2 VM table entry, with L2_VMTE.VALID remaining to be 0.
2. Software writes to IRS_VMAP_VMR.
3. The IRS performs a write to the level 2 VM table entry that updates L2_VMTE.VALID from 0 to 1.
4. The IRS reports that the effects of the write to IRS_VMAP_VMR are complete.

See 3.3 *Coherency considerations for IRS data structures* for more information about ensuring that memory accesses are coherent between the IRS and PE.

$I_{PYJLW}$ When a VM becomes valid, the following fields are interpreted by the IRS Domain:

- L2_VMTE.VMD_ADDR
- L2_VMTE.VPE_ID_BITS
- L2_VMTE.VPET_ADDR
- L2_VMTE.SPI_IST_VALID
- L2_VMTE.SPI_IST_VALID

If the L2_VMTE.LPI_IST_VALID field is 1 when the VM becomes valid, the following fields are interpreted by the IRS Domain:

- L2_VMTE.LPI_ID_BITS
- L2_VMTE.LPI_IST_STRUCTURE
- L2_VMTE.LPI_IST_ADDR
- L2_VMTE.LPI_ISTSZ

If the L2_VMTE.SPI_IST_VALID field is 1 when the VM becomes valid, the following fields are interpreted by the IRS Domain:

- L2_VMTE.SPI_ID_BITS
- L2_VMTE.SPI_IST_STRUCTURE
- L2_VMTE.SPI_IST_ADDR
- L2_VMTE.SPI_ISTSZ

See 4.9.2 *The VPE table* for information about valid entries in the VPE table when a VM becomes valid.

$R_{BYJYM}$ For a write to a valid level 2 VM table entry, all of the following are true:

- If L2_VMTE.LPI_IST_VALID is 0, the following fields are permitted to be updated:
  - L2_VMTE.LPI_ID_BITS.
  - L2_VMTE.LPI_IST_STRUCTURE.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

  - L2_VMTE.LPI_IST_ADDR.
  - L2_VMTE.LPI_ISTSZ.
- If L2_VMTE.SPI_IST_VALID is 0, the following fields are permitted to be updated:
  - L2_VMTE.SPI_ID_BITS.
  - L2_VMTE.SPI_IST_STRUCTURE.
  - L2_VMTE.SPI_IST_ADDR.
  - L2_VMTE.SPI_ISTSZ.

Updating any other field in a valid level 2 VM table entry results in UNPREDICTABLE behavior.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the IRS Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$I_{HQDFR}$ See 4.10.1 *Virtual LPIs* and 4.10.2 *Virtual SPIs* for information about how the virtual ISTs become valid and the corresponding L2_VMTE.LPI_IST_VALID and L2_VMTE.SPI_IST_VALID are updated for a valid VM.

$R_{LDKSX}$ When a level 2 VM table is valid, the IRS behavior is UNPREDICTABLE if there is a write to the level 2 VM table entry when IRS_VMT_STATUSR.IDLE is 0 because of a write to any of the following registers:

- IRS_VMAP_VMR.
- IRS_VMAP_VPER.
- IRS_VMAP_VISTR.
- IRS_VMAP_L2_VISTR.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{FFCKZ}$ If a write occurs to any of the following registers when IRS_VMT_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE whether any effects of any of the writes complete successfully:

- IRS_VMAP_VMR.
- IRS_VMAP_VPER.
- IRS_VMAP_VISTR.
- IRS_VMAP_L2_VISTR.

$D_{BTKBB}$ A VM *becomes invalid* in any of the following scenarios:

- A write to IRS_VMAP_VMR makes the VM invalid.
- The VM table becomes invalid.

$R_{KFTFZ}$ To recover from UNPREDICTABLE behavior resulting from an incorrect write to a valid level 2 VM table entry, the VM is made invalid by a write to IRS_VMAP_VMR that sets IRS_VMAP_VMR.U to 1.

$R_{XXVLZ}$ If a VM becomes invalid and a VPE in the VM is resident on a PE, the resident VPE is treated as belonging to an invalid VM and all of the following are true:

- If a virtual candidate HPPI was selected for the VPE, one of the following is true:
  - The virtual candidate HPPI is acknowledged in finite time.
  - The virtual candidate HPPI is not acknowledged.
- GIC System instructions operating in the Virtual Interrupt Domain and specify the VM are treated as specifying an invalid VM.
- Doorbell requests from the PE where the VPE is resident are IGNORED.

If the VM with the same VM ID subsequently becomes valid, then it's CONSTRAINED UNPREDICTABLE whether the resident VPE is still treated as being resident and belonging to the new VM, or is treated as belonging to an invalid VM.

$S_{WKTGN}$ Software can make a VM invalid and reclaim all resources related to the VM by performing the following sequence:

1. Make all VPEs non-resident.
2. Unmap and invalidate all ITS mappings that target the VM.
3. Unassign all SPIs assigned to the VM.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

4. Make the VM invalid by writing to IRS_VMAP_VMR.

#### 4.9.1.4 The VM descriptor

$I_{ZVXTY}$ An implementation may require a VM descriptor for each VM.

Whether a VM descriptor is required is reported by IRS_IDR3.VMD.

$I_{QYYFH}$ The VM descriptor is memory provided by software to the IRS Domain to manage the VM. The content and usage of the VM descriptor is IMPLEMENTATION DEFINED. The VM descriptor cannot be moved once the VM is made valid.

$I_{YBLTD}$ If a VM descriptor is required, a valid L2_VMTE contains the base addresses of the VM descriptor.

$R_{BWKBG}$ If a VM descriptor is required, when a VM becomes valid, the memory pointed to by L2_VMTE.VMD_ADDR is expected to contain all zeros.

If the VM descriptor does not contain all zeros, all of the following are true:

- It is CONSTRAINED UNPREDICTABLE whether the IRS Domain selects a target VPE for virtual 1ofN interrupts for the VM.
- The VM configuration, including the doorbell settings, is reset to UNKNOWN values.

$R_{HQPRD}$ The format and content of the VM descriptor is IMPLEMENTATION DEFINED.

$I_{BBCFD}$ The size of the VM descriptor is reported by IRS_IDR3.VMDW.

$R_{VNVHD}$ Whether a VM descriptor is required, and its size, is the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

Whether a VM descriptor is required, and its size, is the same across all IRSs.

#### 4.9.1.5 Example VM table structures

$I_{BSPBV}$ The size of a level 1 VMTE is 8 bytes and the size of a level 2 VMTE is 32 bytes.

Table 4.4 shows some example VM table structures.

$S_{VKKQW}$ The L1 size and L2 size columns in Table 4.4 indicate the required maximum physically contiguous allocation by software for the configuration.

**Table 4.4: Example VM table structures**

| STRUCTURE | VM_ID_BITS | L1 size | L2 size | Maximum number of VMs |
|---|---|---|---|---|
| Linear | 7 | - | 4 KB | 128 |
| Linear | 11 | - | 64 KB | 2,048 |
| Linear | 15 | - | 1 MB | 32,768 |
| Linear | 16 | - | 2 MB | 65,536 |
| 2-level | 16 | 4 KB | 4 KB | 65,536 |

### 4.9.2 The VPE table

$I_{BGKFQ}$ The VPE table contains the configuration of the VPEs defined for a VM.

For each VPE, the VPE table contains pointers to other data structures associated with that VPE.

In a multi-IRS system, the VPE table is shared between all IRSs for the IRS Domain.

The VPE table is not permitted to be shared across multiple VMs.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$D_{PFHDR}$ The VPE table uses a linear structure.

$D_{MPMBR}$ The base address of the VPE table specifies the location of the VPE table.

$I_{WXRZV}$ L2_VMTE.VPE_ID_BITS selects maximum number of VPEs for the VM.

$S_{PLJRT}$ When a VM is made valid, it is configured to support a maximum number of VPEs. This value cannot be modified once the VM is made valid. If more VPEs are required than the maximum configured in the L2_VMTE.VPE_ID_BITS field, then the VM can be made invalid and a new VM can be made valid with a larger maximum number of VPEs.

---

**Note**

L2_VMTE.VPE_ID_BITS specifies the maximum number of entries in the VPE table. For some of those entries, VPETE.VALID may be 0, and therefore the number of valid VPEs in the VM may be smaller than the value specified by L2_VMTE.VPE_ID_BITS.

---

$I_{DRHQP}$ The VPE table is indexed using a VPE ID ranging from 0 to (2^L2_VMTE.VPE_ID_BITS) - 1.

$R_{WSDXY}$ When an IRS accesses the VPE table for a VM, the IRS does not access any memory location derived from a VPE ID which is outside the configured VPE ID range for the VM.

This includes situations where the VPE ID is programmed in the parameters to a GIC System instruction, written to a GIC System register, or the VPE ID is stored in a virtualization table structure entry.

$I_{QCQBD}$ The maximum supported VPE ID is reported by IRS_VPE_ID_BITS.

$R_{HBNVF}$ The number of supported VPE ID bits is the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

The number of supported VPE ID bits is the same across all IRSs.

$R_{NQDMK}$ For virtual interrupts, when an Interrupt Affinity ID is specified, it is interpreted as the VPE ID.

$S_{RMDNH}$ A virtual Interrupt Affinity ID is interpreted as the VPE ID for virtual interrupts and used directly to index into the VPE table.

Arm expects that hypervisor software at EL2 presents a virtual IRS with an emulated virtual IRS_PE_SELR register. The value written to the emulated IRS_PE_SELR.IAFFID field should select the emulated configuration registers corresponding to the VPE with the corresponding IAFFID and VPE ID.

#### 4.9.2.1 The VPE table base address and configuration

$I_{CFPXL}$ The base address of the VPE table for a VM is stored in the corresponding L2_VMTE.VPET_ADDR.

See 4.9.1.3 *VM management* for more information.

$I_{LTQZN}$ A VPE table becomes valid when its VM becomes valid.

When a VPE table becomes valid, the VPE table can contain all invalid entries, a combination of valid and invalid entries, or all valid entries.

$R_{PSYQV}$ When VPETE.VALID is 1, the IRS may speculatively access any memory location derived from the address and configuration data in the VPE table entry.

See 4.12 *IRS memory access rules* for more information.

#### 4.9.2.2 VPE management

$D_{BKCHX}$ A VPE for a VM, and the corresponding VPE table entry, are considered *valid* when all of the following are true:

- The VM is valid.
- The corresponding VPETE.VALID is 1.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Otherwise, the VPE and the corresponding VPE table entry are considered *invalid*.

$R_{JKDCN}$ If a VPE ID is outside the configured VPE ID range for the VM, the VPE ID is treated as specifying an invalid VPE.

$G_{QNPPW}$ The architecture supports multiple ways to create a VM:

- A VM can be made valid without containing any valid VPEs, and each VPE is subsequently made valid.
- A VM can be made valid with pre-configured valid VPEs to reduce the overhead of setting up a new VM.

$D_{HNRJX}$ A VPE *becomes valid* in any of the following scenarios:

- A write to IRS_VMAP_VPER makes the VPE valid for an already valid VM.
- A VM becomes valid and VPETE.VALID is 1 for the corresponding VPE table entry for the VM.

$R_{LSYNH}$ When a VPE table is valid, if software writes to an invalid VPE table entry and sets VPETE.VALID to 1, the behavior of the IRS Domain is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{XRNSL}$ When a write to IRS_VMAP_VPER makes the corresponding VPE valid, all of the following are true:

- The IRS generates the following Effects:
  - A Register Write Effect $E_1$ to IRS_VMT_STATUSR which sets the IDLE field to 0.
  - A Memory Write Effect $E_2$ to the Location corresponding to the Level 1 VM table entry which updates VPETE.VALID from 0 to 1.
  - A Register Write Effect $E_3$ to IRS_VMT_STATUSR which sets the IDLE field to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_1$ to the Memory Write Effect $E_2$.
- There is an Intrinsic order dependency from the Memory Write Effect $E_2$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_2$ is Ordered-before $E_4$.

$I_{SBDHK}$ When a VPE table is valid, a VPE is made valid by the following sequence:

1. Software initializes the corresponding VPE table entry, with VPETE.VALID remaining to be 0.
2. Software writes to IRS_VMAP_VPER.
3. The IRS performs a write to the VPE table entry that updates VPETE.VALID from 0 to 1.
4. The IRS reports that the effects of the write to IRS_VMAP_VPER are complete.

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

$I_{TTRHV}$ When the VPE becomes valid, the corresponding VPETE.VPED_ADDR specifies the location of the VPE descriptor.

See 4.9.2.3 *The VPE descriptor* for more information.

$R_{YLKWF}$ When a VPE table is valid, the IRS behavior is UNPREDICTABLE if there is a write to the VPE table entry when IRS_VMT_STATUSR.IDLE is 0 because of a write to IRS_VMAP_VPER.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$R_{LJJVP}$ If a write updates any field in a valid VPE table entry, the IRS Domain behavior is UNPREDICTABLE.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the IRS Domain or in an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

$I_{LVSWT}$ The architecture does not support making an individual VPE invalid without making the corresponding VM invalid.

$R_{PKQLK}$ To recover from UNPREDICTABLE behavior resulting from an incorrect write to a VPE table entry, the VM is made invalid.

$D_{VDSVN}$ A VPE *becomes invalid* when the corresponding VM becomes invalid.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 4.9.2.3 The VPE descriptor

$I_{YWLWM}$ The VPE descriptor is memory provided by software to the IRSs to manage the VPE. The content and usage of the VPE descriptor is IMPLEMENTATION DEFINED. The VPE descriptor cannot be moved once the VPE is created, for as long as the VM remains valid.

$R_{DLKTX}$ When a VPE becomes valid, the memory pointed to by VPETE.VPED_ADDR is expected to contain all zeros.

If the VPE descriptor does not contain all zeros, all of the following are true:

- It is CONSTRAINED UNPREDICTABLE whether the IRS Domain selects a virtual candidate HPPI for the VPE.
- The VPE configuration, including the doorbell settings, is reset to UNKNOWN values.

$R_{DHCDH}$ The format and content of the VPE descriptor is IMPLEMENTATION DEFINED.

$I_{QPDNG}$ The size of the VPE descriptor is reported by IRS_IDR4.VPED_SZ.

$R_{TJMHJ}$ The size of the VPE descriptor is the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

The size of the VPE descriptor is the same across all IRSs.

### 4.9.2.4 Example VPE table structures

$I_{LFWWJ}$ The size of a VPE table entry is 8 bytes.

Table 4.5 shows some example VPE table structures.

The VP_ID_BITS column represents the value stored in the L2_VMTR.VPE_ID_BITS field for the VM containing the VPE table.

**Table 4.5: Example VPE table sizes**

| VPE_ID_BITS | VPE table size | Maximum number of VPEs |
|---|---|---|
| 2 | 32 B | 4 |
| 3 | 64 B | 8 |
| 4 | 128 B | 16 |
| 7 | 1 KB | 128 |
| 9 | 4 KB | 512 |
| 14 | 128 KB | 16,384 |
| 16 | 512 KB | 65,536 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 4.10 Virtual interrupts

$D_{YJQKX}$ *Virtual interrupts* are interrupts that belong to a VM.

$R_{XKHNL}$ A virtual interrupt is only permitted to be selected as the candidate HPPI for the Virtual Interrupt Domain.

$I_{BNPQR}$ The IRS supports the following types of virtual interrupts for VMs:

- Virtual LPIs.
- Virtual SPIs.

Both types of virtual interrupts are stored in virtual ISTs.

$I_{ZDNVV}$ Virtual interrupts are routed to a target VPE and selected as the virtual candidate HPPI for the PE where the target VPE is resident.

See 4.10.6 *VPE residency* and 4.10.3 *Virtual interrupt routing* for mor[illegible]iformat[illegible]

$R_{KPKLS}$ IRS support for virtualization is optional.

$I_{FYJLC}$ IRS_IDR0.VIRT indicates whether virtualization is supported.

$R_{DGHCV}$ IRS_IDR0.VIRT is 0 for the EL3 Interrupt Domain.

$R_{KDSPG}$ When at least one PE connected to an IRS implements EL[illegible] IR[illegible]_IDR0.VIRT is 1 for all IRSs for all Interrupt Domains other than the EL3 Interrupt Domain.

When none of the PEs in the system implemen[illegible]EL2, IR[illegible]_IDR0.V[illegible]s 0 for all IRSs for all Interrupt Domains.

$R_{JJPQJ}$ The value of IRS_IDR0.VIRT is the same fo[illegible] IRS[illegible]e Interrupt Domain in the system.

$R_{TQFCT}$ When virtualization is not supported, the [illegible]S does [illegible]t select any virtual candidate HPPIs to the PEs and any virtual interrupts signaled to the IRS are [illegible]NORED[illegible]

$R_{TZBQY}$ The number of Priority bits su[illegible]ed for virtua[illegible]errupts corresponds to the number of Priority bits supported in the Interrupt Domain wher[illegible]ne VM[illegible] defined.

See 4.6 *Interrupt con[illegible]uration and sta[illegible]* for more information about the number of supported Priority bits.

### 4.10.1 Virtual LPIs

$D_{JXYLG}$ The virtual [illegible]PIs for a[illegible]M are st[illegible]d in the *virtual LPI IST* for the VM.

$I_{RMNHL}$ The vi[illegible]ual LPI[illegible]ST is allo[illegible]d separately for each VM.

The [illegible]tual L[illegible]VM is shared across all IRSs in the system.

A virtua[illegible]PI IST is not permitted to be shared across multiple VMs.

$I_{WPHXH}$ The number o[illegible]irtual LPIs in a VM is configured in L2_VMTE.LPI_ID_BITS.

Software allocates and provides the virtual LPI IST for a VM by writing a valid address to L2_VMTE.LPI_IST_ADDR and writing 1 to L2_VMTE.LPI_IST_VALID.

$I_{WPZYD}$ The virtual LPI IST is indexed using the INTID.ID in the range from 0 to (2 ^ L2_VMTE.LPI_ID_BITS) - 1.

$R_{CJYYQ}$ When an IRS accesses the virtual LPI IST for a VM, the IRS does not access any memory location derived from an INTID which is outside the configured virtual LPI range for the VM.

This includes situations where the INTID is programmed in the parameters to a GIC System instruction or the INTID is stored in the IST metadata areas.

$R_{DJZRM}$ A virtual LPI specified by an INTID for a VM specified by a VM ID is reachable when all of the following are true:

- The VM ID is valid.
- The INTID is within the configured virtual LPI range for the VM.
- The virtual LPI IST for the VM is valid.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- There is a level 2 IST entry corresponding to the INTID in the virtual LPI IST.

Otherwise, the virtual LPI is unreachable.

$I_{GQQSW}$ In the virtual LPI IST context, the base address of the virtual LPI IST for a VM is stored in L2_VMTE.LPI_IST_ADDR.

$I_{JKPTM}$ In the virtual LPI IST context, L2_VMTE.LPI_IST_VALID determines if the virtual LPI IST is valid for a VM.

$D_{WDHDY}$ A virtual LPI IST for a VM *becomes valid* in one of the following scenarios:

- The VM becomes valid and L2_VMTE.LPI_IST_VALID is 1 in the corresponding level 2 VM table entry.
- The VM is valid and a write to IRS_VMAP_VISTR makes the virtual LPI IST valid.

See 4.7 *The interrupt state table (IST)* for more information about an IST becoming valid and invalid.

$R_{XKGXC}$ When a write to IRS_VMAP_VISTR makes the corresponding LPI IST f[illegible] VM valid or invalid, all of the following are true:

- The IRS generates the following Effects:
  - A Register Write Effect $E_1$ to IRS_VMT_STATUSR [illegible]ich se[illegible]he IDLE f[illegible]d to 0.
  - A Memory Write Effect $E_2$ to the Location corresp[illegible]ding to the [illegible]el 2 V[illegible] table entry which updates L2_VMTE.LPI_IST_VALID.
  - A Register Write Effect $E_3$ to IRS_VMT_STA[illegible]SR wh[illegible]n sets the IDLE field to 1.
- There is an Intrinsic order dependency from the Regist[illegible]rite Effec[illegible]$_1$ to the Memory Write Effect $E_2$.
- There is an Intrinsic order dependency from [illegible] Memory [illegible]ite Ef[illegible]ct $E_2$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read E[illegible]ect $E_4$ w[illegible]ch Reads-f[illegible]om the Register Write Effect $E_3$, then $E_2$ is Ordered-before $E_4$.

$I_{ZTSBX}$ When a VM is valid, the virtual LPI IST [illegible] made v[illegible]d by the following sequence:

1. Software configures the v[illegible]ual LPI IS[illegible] by wr[illegible]g to the corresponding level 2 VM table entry, with L2_VMTE.LPI_IST_V[illegible] remaining to[illegible]0.
2. Software writes to IR[illegible]_VMA[illegible]VISTR.
3. The IRS performs [illegible]write to the l[illegible]l 2 VM table entry that updates L2_VMTE.LPI_IST_VALID from 0 to 1.
4. The IRS report[illegible]hat the effects of t[illegible]write to IRS_VMAP_VISTR are complete.

See 3.3 *Coher[illegible]y consi[illegible]ations for GIC data structures* for more information about ensuring that memory accesses are cohe[illegible]b[illegible]ween the I[illegible]S and a PE.

See 4.9 *[illegible] vi[illegible]manag[illegible]ent* f[illegible]more information about which fields are permitted to be updated when the VM is valid.

$I_{BHBYF}$ See [illegible].1.3 *[illegible]l ma[illegible]ent* for more information about updates to a level 2 VM table entry and the use of the virtual i[illegible] registers.

$S_{KBXNL}$ The architec[illegible]e supports the following ways to configure virtual LPIs for a VM:

- Software can write a valid IST pointer and IST configuration data to the level 2 VM table entry after the VM is made valid and write to the map virtual IST register.
- Software can provide a valid IST pointer in the level 2 VM table entry when the VM is made valid.

Adding an IST to add virtual LPI support for a VM can, for example, be done in response to a guest operating system booting and configuring a virtual IRS emulated by software.

Providing a valid IST pointer at the time the VM is made valid can be used to support VM migration, or to support suspend and resume of VMs.

$I_{SYDWQ}$ When a virtual LPI IST becomes valid, the following fields are interpreted by the IRS Domain:

- L2_VMTE.LPI_ID_BITS
- L2_VMTE.LPI_IST_STRUCTURE
- L2_VMTE.LPI_IST_ADDR
- L2_VMTE.LPI_ISTSZ

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{LTXVZ}$ When IRS_IDR2.IST_levels is 0, L2_VMTE.LPI_IST_STRUCTURE is treated as 0.

$I_{NMFLD}$ IRS_IDR2.{MIN_LPI_ID_BITS,ID_BITS} report the range of valid values for L2_VMTE.LPI_ID_BITS.

In a multi-IRS system, when the virtual LPI IST is valid for a VM, and the VM table is valid, all IRSs are permitted to access the virtual LPI IST for any reason, including speculative reads.

See 4.9.1.1 *The VM table base address and configuration registers* for more information.

$R_{TDTCJ}$ When the virtual LPI IST is invalid for a VM, the IRS Domain does not access the virtual LPI IST.

$D_{PCVXZ}$ A virtual LPI IST for a VM *becomes invalid* in one of the following scenarios:

- The VM becomes invalid.
- The VM is valid and a write to IRS_VMAP_VISTR makes the virtual LPI IST invalid.

See 4.7 *The interrupt state table (IST)* for more information about an IST becoming valid and invalid.

$I_{QFTNB}$ When a VM is valid, the virtual LPI IST is made invalid by the following sequence:

1. Software writes to IRS_VMAP_VISTR.
2. The IRS performs a write to the level 2 VM table entry that updates L2_VMTE.LPI_IST_VALID from 1 to 0.
3. The IRS reports that the effects of the write to IRS_VMAP_VISTR are complete.

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

$R_{CXWBT}$ If a virtual LPI IST for a VM becomes invalid, a VPE in the VM is resident on a PE, and a virtual LPI is the candidate HPPI for the Virtual Interrupt Domain, one of the following is true:

- The candidate HPPI is acknowledged by the PE before the IST becomes invalid.
- The candidate HPPI is not acknowledged and is no longer selected as the candidate HPPI for the VPE.

$R_{KYMTZ}$ When a virtual LPI IST becomes invalid, if there are Accepted events for virtual LPIs targeting the VM that are not yet processed, all of the following are true:

- The events are dropped and have no effect on any interrupt.
- If software error reporting is supported, the IS is permitted to report an error.

$S_{BWSKR}$ If hypervisor software wishes to update the configuration of a virtual LPI IST, for example as the result of guest software programming emulated IRS registers, software should perform the following sequence:

1. Make the virtual LPI IST invalid.
2. Update the relevant fields in the level 2 VM table entry.
3. Make the virtual LPI IST valid.

$I_{PCNMM}$ In the virtual LPI IST context, when the virtual LPI IST is valid and uses a 2-level structure, a virtual level 2 IST is made valid by writing to IRS_VMAP_L2_VISTR.

The write specifies that the map applies to the LPI IST by setting IRS_VMAP_L2_VISTR.TYPE to LPI.

The effects of the write are complete when IRS_VMT_STATUSR.IDLE is 1.

See 4.7.1 *Level 2 IST management* for more information.

$I_{SBDCY}$ See 4.9.1.3 *VM management* for more information about the restrictions for the use of IRS_VMAP_L2_VIST concurrently with updates to the virtualization data structures for the VM and the use of other virtual map registers across IRSs.

See also:

- 4.7 *The interrupt state table (IST)*
- 4.9 *Virtualization data structures*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 4.10.2 Virtual SPIs

$D_{JSJKN}$ The virtual SPIs for a VM are stored in the *virtual SPI IST* for the VM.

$I_{YZPNL}$ The virtual SPI IST is allocated separately for each VM.

The virtual SPI IST for a VM is shared across all IRSs in the system.

A virtual SPI IST is not permitted to be shared across multiple VMs.

$I_{PVHMD}$ The number of virtual SPIs in a VM is configured in L2_VMTE.SPI_ID_BITS.

Software allocates and provides the virtual SPI IST for a VM by writing a valid address to L2_VMTE.SPI_IST_ADDR and writing 1 to L2_VMTE.SPI_IST_VALID.

$I_{DXXHY}$ The virtual SPI IST is indexed using the INTID.ID in the range from 0 through (2 ^ L2_VMTE.SPI_ID_BITS) - 1.

$R_{JQZKY}$ When an IRS accesses the virtual SPI IST for a VM, the IRS does not acc[illegible] any m[illegible]ory location derived from an INTID which is outside the configured virtual SPI range for the VM.

This includes situations where the INTID is programmed in the parame[illegible]s of a GIC S[illegible]stem instruction executed on a PE or the INTID is stored in the IST metadata areas.

$R_{MYLWH}$ A virtual SPI specified by an INTID for a VM specified by [illegible]VM ID i[illegible]eachable[illegible]en all of the following are true:

- The VM is valid.
- The virtual SPI IST for the VM is valid.
- The INTID is within the configured virtu[illegible]SPI ra[illegible]e for the [illegible]M.
- There is a level 2 IST entry correspond[illegible]g to the [illegible]TID in the virtual SPI IST.

Otherwise, the virtual SPI is unreachab[illegible]

$I_{HRDGP}$ In the virtual SPI IST contex[illegible] the [illegible]e addr[illegible]s of the virtual SPI IST for a VM is stored in L2_VMTE.SPI_IST_ADDR.

$I_{CSNJM}$ In the virtual SPI IST conte[illegible], L2_[illegible]MTE.SPI_IS[illegible]_VALID determines if the virtual SPI IST is valid for a VM.

$D_{BKKXD}$ A virtual SPI IST for a [illegible]M *becomes va[illegible]d* in one of the following scenarios:

- The VM bec[illegible]es valid[illegible]nd L2_VMTE.SPI_IST_VALID is 1 in the corresponding level 2 VM table entry.
- The VM is [illegible]lid and [illegible]write to IRS_VMAP_VISTR makes the virtual SPI IST valid.

See 4.7 *The interrup[illegible]ate table (IST)* for more information about an IST becoming valid and invalid.

$R_{MLTBD}$ When [illegible] write t[illegible]RS_VM[illegible]_VISTR makes the corresponding SPI IST for the VM valid or invalid, all of the foll[illegible]ing are t[illegible]e:

- T[illegible]S generates the following Effects:
  - [illegible]Register Write Effect $E_1$ to IRS_VMT_STATUSR which sets the IDLE field to 0.
  - A [illegible]mory Write Effect $E_2$ to the Location corresponding to the level 2 VM table entry which updates L2_VMTE.SPI_IST_VALID.
  - A Register Write Effect $E_3$ to IRS_VMT_STATUSR which sets the IDLE field to 1.
- There is an Intrinsic order dependency from the Register Write Effect $E_1$ to the Memory Write Effect $E_2$.
- There is an Intrinsic order dependency from the Memory Write Effect $E_2$ to the Register Write Effect $E_3$.

This means that if there is a Memory Read Effect $E_4$ which Reads-from the Register Write Effect $E_3$, then $E_2$ is Ordered-before $E_4$.

$I_{CDTQL}$ When a VM is valid, the virtual SPI IST is made valid by the following sequence:

1. Software configures the virtual SPI IST by writing to the corresponding level 2 VM table entry, with L2_VMTE.SPI_IST_VALID remaining to be 0.
2. Software writes to IRS_VMAP_VISTR.
3. The IRS performs a write to the level 2 VM table entry that updates L2_VMTE.SPI_IST_VALID from 0 to 1.
4. The IRS reports that the effects of the write to IRS_VMAP_VISTR are complete.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

See 4.9.1.3 *VM management* for more information about which fields are permitted to be updated when the VM is valid.

$S_{LHQJY}$ Arm expects that hypervisor software provides a valid virtual SPI when a VM is made valid with a fixed number of SPIs for the lifetime of the VM. This is analogous to how SPIs work on a physical system.

$I_{ZZGXF}$ When a virtual SPI IST becomes valid, the following fields are interpreted by the IRS Domain:

- L2_VMTE.SPI_ID_BITS
- L2_VMTE.SPI_IST_STRUCTURE
- L2_VMTE.SPI_IST_ADDR
- L2_VMTE.SPI_ISTSZ

$I_{ZKTZX}$ When IRS_IDR2.IST_levels is 0, L2_VMTE.SPI_IST_STRUCTURE is treated as

$I_{RGNRJ}$ IRS_IDR2.ID_BITS reports the maximum value for L2_VMTE.SPI_ BITS.

$I_{TWRCJ}$ See 4.9.1.3 *VM management* for more information about updates to a level 2 VM table entry and the use of the virtual map registers.

In a multi-IRS system, when the virtual SPI IST is valid for a VM, and the VM table is valid, all IRSs are permitted to access the virtual SPI IST for any reason, including speculative reads.

See 4.9.1.1 *The VM table base address and configuration registers* for more information.

$R_{KYYYR}$ When the virtual SPI IST is invalid for a VM, the IRS in does not access the virtual SPI IST.

$D_{FYKVT}$ A virtual SPI IST for a VM *becomes invalid* in one of the following scenarios:

- The VM becomes invalid.
- The VM is valid and a write to IRS_VMAP_VISTR makes the virtual SPI IST invalid.

See 4.7 *The interrupt state table (IST)* for more information about an IST becoming valid and invalid.

$I_{PZQWR}$ When a VM is valid, the virtual SPI IST is made invalid by the following sequence:

1. Software writes to IRS_VMAP_VISTR.
2. The IRS performs a write to the level 2 VM table entry that updates L2_VMTE.SPI_IST_VALID from 1 to 0.
3. The IRS reports that the effects of the write to IRS_VMAP_VISTR are complete.

See 3.3 *Coherency considerations for GIC data structures* for more information about ensuring that memory accesses are coherent between the IRS and a PE.

$R_{JZQCG}$ If a virtual SPI IST for a VM becomes invalid, a VPE in the VM is resident on a PE, and a virtual SPI is the candidate HPPI for the Virtual Interrupt Domain, one of the following is true:

- The candidate HPPI is acknowledged by the PE before the IST becomes invalid.
- The candidate HPPI is not acknowledged and is no longer selected as the candidate HPPI for the VPE.

$R_{SGLWK}$ When a virtual SPI IST becomes invalid, if there are Accepted events for virtual SPIs targeting the VM that are not yet processed, all of the following are true:

- The events are dropped and have no effect on any interrupt.
- If software error reporting is supported, the IRS is permitted to report an error.

$I_{GVQDH}$ In the virtual SPI IST context, when the virtual SPI IST is valid and uses a 2-level structure, a virtual level 2 IST is made valid by writing to IRS_VMAP_L2_VISTR.

The write specifies that the map applies to the SPI IST by setting IRS_VMAP_L2_VISTR.TYPE to SPI.

The effects of the write are complete when IRS_VMT_STATUSR.IDLE is 1.

See 4.7.1 *Level 2 IST management* for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- 4.7 *The interrupt state table (IST)*
- 4.9 *Virtualization data structures*

#### 4.10.2.1 Assigning physical SPIs to VMs

$D_{WGVHQ}$ A physical SPI is *assigned to a VM* when all of the following are true:

- A write sets IRS_SPI_VMR.VIRT to 1.
- The write specifies a valid VM.
- The write completes successfully as indicated by IRS_SPI_STATUSR.{V,IDLE} being {1,1}.

$I_{KJJNL}$ Assigning SPIs to a VM is only supported when IRS_IDR0.VIRT is 1.

When IRS_IDR0.VIRT is 0, IRS_SPI_VMR is RAZ/WI.

$I_{BSYVR}$ When an SPI is statically assigned to the EL3 Interrupt Domain, it is not possible to assign the SPI to a VM.

Access to IRS_SPI_VMR is RAZ/WI when the SPI is assigned to the EL3 Interrupt Domain.

$R_{PYSML}$ For each SPI that is not statically assigned to the EL3 Interrupt Domain, it is IMPLEMENTATION DEFINED whether the SPI can be assigned to a VM.

When support for assigning an SPI to a VM is implemented, the SPI is assigned to a VM in the Interrupt Domain where the SPI is assigned.

When support for assigning an SPI to a VM is not implemented, access to IRS_SPI_VMR, when that SPI is selected, is RAZ/WI.

$I_{LKYQR}$ Arm recommends that when IRS_IDR0.VIRT is 1, for all SPIs that are connected to an interrupt input signal, that support for assigning the SPI to a VM is implemented.

$R_{NZFQQ}$ When an SPI is assigned to a VM, all of the following are true for the physical SPI:

- The SPI is not considered for physical candidate HPPI.
- The SPI is unreachable.

The effects on virtual SPIs to which physical SPIs are assigned are described in 4.10.4 *Virtual interrupt signaling*.

$R_{NSTHN}$ If a physical SPI is assigned to a VM, GIC System instructions that update the Pending state of the physical SPI are IGNORED.

$R_{VYNSP}$ When a physical SPI is assigned to a virtual SPI, the virtual SPI INTID is reset to its reset state.

If virtual 1ofN interrupt routing is not supported, the reset state of Routing mode for a virtual SPI is Targeted.

When a physical SPI is unassigned from a virtual SPI, all of the following are true:

- The physical SPI INTID is reset to its reset state.
- The virtual SPI INTID is reset to its reset state.

$I_{BGSSV}$ The Trigger mode of an SPI determines the type of the events that the IRS generates for the SPI, regardless of whether the SPI is assigned to a VM.

$R_{HHBSB}$ If IRS_SPI_VMR.VM_ID does not specify a valid VM for an SPI written to IRS_SPI_SELR, it is CONSTRAINED UNPREDICTABLE whether the SPI is treated as not being assigned to any VM, or assigned to an invalid VM.

If the SPI is treated as being assigned to an invalid VM, all of the following are true:

- The SPI is not considered for the physical or virtual candidate HPPI for any PE.
- The SPI is unreachable.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 4.10.3 Virtual interrupt routing

$D_{PMQVK}$ The Routing mode of a virtual interrupt determines how the target VPE is selected for that interrupt in the following ways:

- If the Routing mode is Targeted, the target VPE is the VPE specified by the interrupt Affinity.
- If the Routing mode is 1ofN, the target VPE is determined dynamically.

$D_{SZXHC}$ When an interrupt Routing mode is Targeted and the interrupt Affinity specifies a VPE, the interrupt is said to be *targeted to* that VPE.

$I_{VSDPM}$ If the Routing mode of a virtual interrupt is Targeted, the interrupt Affinity specifies the target VPE. If the Routing mode of a virtual interrupt is 1ofN, the interrupt Affinity is IMPLEMENTATION DEFINED.

$R_{BFGYB}$ IRS support for virtual 1ofN interrupt routing is optional.

$I_{JQMTB}$ IRS_IDR0.VIRT_ONE_N reports whether virtual 1ofN interrupt routing is supported.

$R_{XLGVV}$ The value of IRS_IDR0.VIRT_ONE_N is the same across all Interrupt Domains, except for the EL3 Interrupt Domain, where VMs are not supported.

The value of IRS_IDR0.VIRT_ONE_N is the same across all IRSs.

$R_{JCSBY}$ If virtual 1ofN interrupt routing is not supported, all of the following are true:

- If a GIC System instruction generates an Interrupt Effect that sets the Routing mode of a virtual interrupt, the Routing mode remains set to Targeted regardless of the value provided in `<Xt>`.
- L2_ISTE.IRM is treated as 0 when a virtual IST becomes valid.

$S_{QQDND}$ When virtual 1ofN interrupt routing is not supported, the IRM field in the value encoded in the `<Xt>` parameter of the `GIC xDAFF` System instruction is RES0 and `GIC xDRCFG` instructions always return 0 in the IRM field in ICC_ICSR_EL1 when requesting the configuration of a virtual interrupt.

This means that virtual 1ofN interrupts are treated as Targeted interrupts and are always delivered to the VPE specified by the interrupt Affinity.

Hypervisor software can support virtual 1ofN interrupt routing in systems where virtual 1ofN interrupt routing is not implemented by emulating the 1ofN behavior.

Hypervisor software can emulate virtual 1ofN by trapping guest access to the configuration of virtual interrupts and maintain a shadow configuration of each interrupt used by the physical IRS and updating the target VPE of the shadow interrupt to emulate a 1ofN selection algorithm.

$R_{TBBTW}$ Each virtual 1ofN interrupt can be acknowledged by at most one VPE.

$R_{FHDFL}$ The storage location for the Affinity of virtual 1ofN interrupts may be used for an IMPLEMENTATION DEFINED purpose to handle routing of virtual 1ofN interrupts, with the restriction that an Affinity value of 0 has no IMPLEMENTATION DEFINED meaning.

$R_{FRMXL}$ An IRS is permitted to update the Affinity of virtual 1ofN interrupts.

$I_{BYXVC}$ When virtual 1ofN is supported, the VPE can be configured to have 1ofN PE selection enabled or disabled by accessing IRS_VPE_CR0.DPS.

$R_{SCWSJ}$ When virtual 1ofN is supported, an IRS does not choose a VPE which has 1ofN PE selection disabled as the target VPE for a 1ofN interrupt.

$R_{XMNZS}$ When virtual 1ofN is supported, the IRS Domain selects a target VPE for a reachable virtual 1ofN interrupt belonging to a valid VM in finite time, when all of the following are true:

- The interrupt is Enabled.
- The interrupt is Pending.
- The interrupt is Inactive.
- At least one IRS is enabled.
- At least one VPE in the VM has 1ofN PE selection enabled.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

The target VPE is selected among the VPEs in the VM that the virtual 1ofN interrupt belongs to.

**Note**

The target VPE selection behavior does not guarantee that the interrupt is selected as the virtual candidate HPPI for a PE. For example, the selected target VPE may be masking the interrupt, the VPE may not be resident, another virtual interrupt may be the virtual candidate HPPI for the target VPE.

$R_{SKVBZ}$ An IRS Domain may select a new target VPE for a virtual 1ofN interrupt at any time.

$R_{QYKSH}$ When virtual 1ofN is supported, the mechanism that an IRS uses to select a target VPE among several possible target VPEs is IMPLEMENTATION DEFINED.

$S_{RBXBB}$ When virtual 1ofN is supported, the VPE 1ofN PE selection enable cont[illegible] may [illegible]sed to support virtualization of IRS_PE_CR0.DPS.

Arm expects that hypervisor software at EL2 traps the guest programm[illegible] of the virtu[illegible] IRS_PE_CR0.DPS field and configures the VPE accordingly.

$I_{DWNMY}$ When IRS_VPE_CR0.DPS is updated for a VPE that is res[illegible]ent on a [illegible], and a [illegible]al 1ofN interrupt is selected as the virtual candidate HPPI for the VPE, one of the follow[illegible] is tr[illegible]:

- The PE acknowledges the interrupt before IR[illegible]PE_ST[illegible]USR.ID[illegible]E is 1.
- The PE does not acknowledge the interru[illegible].

See 4.10.4 *Virtual interrupt signaling* for m[illegible] inform[illegible].

### 4.10.4 Virtual interrupt signaling

$R_{GMMHX}$ The Pending state of a virtual L[illegible] identified b[illegible] INTID and a VM ID is updated when all of the following are true:

- The virtual LPI as [illegible]entified by t[illegible] INTID is reachable.
- The VM as ide[illegible]ied by the VM ID[illegible] valid.
- Any of the f[illegible]owing o[illegible]ur:
  - An ITS [illegible]enerat[illegible] an interrupt event that specifies the VM ID and INTID.
  - A GIC Sy[illegible] instructio[illegible] generates an Interrupt Write Effect that updates the Pending state of the virtual [illegible]PI.

Oth[illegible]wise, the [illegible]rtual LPI P[illegible]nding state is not updated.

$I_{YDCMB}$ The IR[illegible] SP[illegible] LPI register does not support signaling virtual interrupts.

$R_{CQTDT}$ The Pendi[illegible]tate of a virtual SPI identified by an INTID and a VM ID is updated when all of the following are true:

- The virtual SPI as identified by the INTID is reachable.
- The VM as identified by the VM ID is valid.
- Any of the following occur:
  - An interrupt connected to an IRS as an SPI is asserted or de-asserted and generates an interrupt event for the virtual SPI because the SPI is assigned to the VM.
  - A GIC System instruction generates an Interrupt Write Effect that updates the Pending state of the virtual SPI.

Otherwise, the virtual SPI Pending state is not updated.

$I_{BZFTG}$ When a virtual interrupt is signaled to the IRS, the IRS updates the Pending state of the virtual interrupt regardless of whether the target VPE is resident or non-resident.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$D_{HCZPD}$ The IRS selects a virtual candidate HPPI for a VPE for an Interrupt Domain, when at least one virtual interrupt meets the virtual *candidate HPPI conditions*:

- The interrupt is Enabled.
- The interrupt is Pending.
- The interrupt is Inactive.
- One of the following is true:
  - The interrupt Routing mode is Targeted and the interrupt Affinity specifies the VPE.
  - The interrupt Routing mode is 1ofN and the VPE is selected as the target VPE for the interrupt.

$R_{BFYMP}$ For a virtual Targeted interrupt whose interrupt Affinity specifies an invalid VPE that is updated from invalid to valid, whether the interrupt is considered for the virtual candidate HPPI selection for the VPE depends on whether there is an update to the following states and configurations for the virtual interrupt:

- Pending.
- Active.
- Enabled.
- Routing mode.
- Affinity.

If there is no update to any of the mentioned states and configurations of the virtual interrupt after the VPE becomes valid, it is CONSTRAINED UNPREDICTABLE whether the interrupt is considered for the virtual candidate HPPI selection for the VPE.

If there is an update to any of the mentioned states and configurations for the virtual interrupt after the VPE becomes valid, the virtual interrupt is considered for the virtual candidate HPPI selection for the VPE.

$R_{CPCCK}$ For a VPE in a VM, all of the following apply to the virtual candidate HPPI selection:

- If at least one interrupt satisfies the virtual candidate HPPI conditions, the IRS selects one of these as the virtual candidate HPPI in finite time.
- If no interrupts satisfy the virtual candidate HPPI conditions, the IRS does not select any interrupt as the virtual candidate HPPI.

$R_{ZDTYM}$ If more than one interrupt satisfies the virtual candidate HPPI conditions for a VPE in a VM, the IRS selects the interrupt with the highest Priority as the virtual candidate HPPI in finite time.

If there is more than one virtual candidate HPPI with the same Priority, it is IMPLEMENTATION DEFINED which of those interrupts is selected as the virtual candidate HPPI.

$R_{VBPZM}$ An IRS selects a virtual candidate HPPI for a connected PE in finite time, when all of the following are true:

- There is a resident VPE on the PE.
- The IRS is enabled for the Physical Interrupt Domain corresponding to the resident VPE.
- There is at least one virtual interrupt that meets the virtual *candidate HPPI conditions* for the VPE.

$I_{WZNJR}$ If the IRS Domain has selected a virtual candidate HPPI for a non-resident VPE, it might generate a VPE doorbell. See 4.10.7 *VPE doorbells* for more information.

$R_{XBNGN}$ When a virtual candidate HPPI that has been selected for a VPE is no longer the virtual candidate HPPI for the VPE, and if the VPE is resident on a PE, one of the following is true:

- The PE acknowledges the old virtual candidate HPPI within finite time.
- The old virtual candidate HPPI is not acknowledged.

$R_{ZFVRC}$ If the IRS updates the configuration of an interrupt that has been selected as the virtual candidate HPPI for the Virtual Interrupt Domain for a PE, and the interrupt configuration data was communicated to the PE, the IRS will select the same or a new virtual interrupt as the virtual candidate HPPI.

The IRS then communicates the selected virtual candidate HPPI to the PE, including any updated interrupt configuration data, in finite time.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

For more information about ordering of such events, see 2.12 *Interrupt ordering model and synchronization requirements* and Chapter B1 *Interrupt ordering litmus tests*.

See also:

- 4.8.4 *Physical interrupt signaling*

### 4.10.5 VPE selection and configuration

$I_{BNNMF}$ Software accesses the configuration and samples the HPPI of a VPE by selecting the VPE and accessing one of the following registers:

- IRS_VPE_CR0.
- IRS_VPE_DBR.
- IRS_VPE_HPPIR.

Software selects a VPE by writing the VM ID and VPE ID to IRS_VPE_SELR.{VM_ID,VPE_ID} and writing 1 to IRS_VPE_SELR.S.

IRS_VPE_STATUSR.IDLE reports whether the effects of the write IRS_VPE_SELR are complete.

IRS_VPE_STATUSR.V reports whether the value written to IRS_VPE_SELR selected a valid VPE.

IRS_VPE_STATUSR.V is updated on a write to any of the following registers:

- IRS_VPE_CR0.
- IRS_VPE_DBR.
- IRS_VPE_HPPIR.
- IRS_VPE_SELR.

$R_{MVDZK}$ If a VPE is selected on more than one IRS in a multi-IRS system, and a write occurs to any of the VPE configuration registers on one IRS while IRS_VPE_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain due to a write to the configuration register, it is CONSTRAINED UNPREDICTABLE which one of the following is true:

- The write does not successfully update the VPE and IRS_VPE_STATUSR.V is 0 on one of the IRSs where a write occurs.
- None of the writes updates the VPE configuration.
- Both of the writes successfully update the VPE configuration in UNKNOWN order.

$R_{RLHSL}$ When a VPE becomes valid, the VPE configuration is reset to the following values:

- If virtual 1ofN is supported, the VPE can receive both Targeted and 1ofN interrupts.
- If virtual 1ofN is not supported, the VPE can only receive Targeted interrupts.
- [illegible] all settings for the VPE.

$I_{MFSZW}$ The configuration of a VPE when it becomes valid is consistent with a read of IRS_VPE_CR0.DPS returning 0, assuming the VPE is selected in IRS_VPE_SELR and there was no write to IRS_VPE_CR0 since the VPE became valid.

$I_{HHWBZ}$ A write to any of the VPE configuration registers is only guaranteed to have successfully completed when IRS_VPE_STATUSR.{V,IDLE} is {1,1}.

If the VPE becomes invalid after the VPE is selected, a write to any of the VPE configuration registers results in IRS_VPE_STATUSR.V being 0.

$I_{RXTXP}$ A read of IRS_VPE_HPPIR returns whether there is a candidate HPPI for the selected VPE.

If there is a candidate HPPI for the selected VPE, IRS_VPE_HPPIR.{TYPE,ID} returns the INTID of the candidate HPPI.

$I_{PNRVY}$ IRS_VPE_HPPIR reports the virtual candidate HPPI for a VPE. When virtual 1ofN interrupt routing is not supported, the reported candidate HPPI is a targeted interrupt. When virtual 1ofN interrupt routing is supported, the reported candidate HPPI may be either a targeted or 1ofN virtual interrupt.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 4.10.6 VPE residency

DYTZXF The IRS tracks, for each VPE, whether the VPE is resident VPE, and if it is resident, which PE it is resident on. If a VPE is not resident on any PE, the VPE is *non-resident*.

ILGVMQ A VPE is made resident or non-resident on a PE as a result of a write to ICH_CONTEXTR_EL2.

RZFXNN If a write to ICH_CONTEXTR_EL2 to make a VPE resident specifies an invalid VPE, the write does not cause a VPE to be considered resident by the IRS.

RJJKJV If a PE requests that a VPE is made resident on a PE when the VPE is already resident on another PE, it is CONSTRAINED UNPREDICTABLE which one of the following applies:

- The request is IGNORED.
- The request completes and all of the following are true:
  - It is CONSTRAINED UNPREDICTABLE which of the PEs are allowed to acknowledge and configure interrupts from the VPE context.
  - It is CONSTRAINED UNPREDICTABLE whether making a VPE non-resident on some or all of the PEs results in the VPE being resident on any PEs or non-resident.

To recover from the CONSTRAINED UNPREDICTABLE behaviour, software makes the VPE non-resident on all PEs where it was previously made resident, and the VPE can subsequently be made resident on a single PE.

See also:

- 4.9 *Virtualization data structures*

### 4.10.7 VPE doorbells

GMGLGB The architecture provides a mechanism to signal a physical interrupt when a virtual interrupt is available for a non-resident VPE. This allows a hypervisor to make scheduling decisions about when to run a VPE.

DJNQYP A *VPE doorbell* is the mechanism that generates a physical interrupt event when the VPE doorbell conditions are met.

RCWZMW A VPE doorbell event is generated for a VPE, when all of the following *VPE doorbell conditions* are true:

- The VPE is not resident on any PE.
- The VPE doorbell settings are valid for the VPE.
- A VPE doorbell is requested for the VPE.
- Any one of the following is true:
  - There is at least one Targeted virtual interrupt for the VPE that is Pending, Inactive, and Enabled.
  - The 1ofN doorbell conditions are met for the VM and the VPE is the 1ofN VPE doorbell target. See 4.10.8 *1ofN doorbells* for more information.
- The interrupt causing the doorbell event to be generated has a priority greater than or equal to the doorbell priority mask.

---

**Note**

The VPE doorbell priority mask is checked both when there is a Targeted virtual interrupt causing the VPE doorbell to be generated, and when the VPE doorbell is generated because the 1ofN doorbell conditions are met.

This means that when the 1ofN doorbell conditions are met, they only cause a VPE doorbell event to be generated when the virtual 1ofN interrupt Priority is greater than or equal to the doorbell priority mask.

---

RPCXCN When a VPE is created, the doorbell settings for the VPE are not valid. The doorbell settings become valid when the VPE is selected using IRS_VPE_SELR and a write to IRS_VPE_DBR occurs.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{KDQNS}$ When support for LPIs is implemented, VPE doorbells generate SET_EDGE targeting physical LPIs in the same Interrupt Domain as where the VPE is defined. Otherwise, the mechanism to signal a VPE doorbell is IMPLEMENTATION DEFINED.

$I_{GBLPY}$ An LPI used for a VPE doorbell can be configured the same way as any other LPI including configuring its Priority, Enabled value, Routing mode, and Affinity when relevant.

$I_{KTNWK}$ The doorbell settings of a VPE can be accessed by selecting the VPE by writing its VM ID and VPE ID to IRS_VPE_SELR and accessing IRS_VPE_DBR.

$I_{MZXXV}$ The interrupt used for the VPE doorbell is programmed by writing to IRS_VPE_DBR.INTID.

$D_{MGQPJ}$ A VPE doorbell is *requested* when IRS_VPE_DBR.REQ_DB is 1.

$I_{CVJVS}$ A VPE doorbell is requested in any of the following ways:

- When software makes a VPE non-resident, it can program whether a VPE doorbell is requested.
- Software can access IRS_VPE_DBR.REQ_DB using the VPE selection and configuration interface on an IRS.

$R_{WTZVF}$ Once a VPE doorbell event is generated, IRS_VPE_DBR.REQ_DB is cleared to 0 and the doorbell is no longer requested for the VPE.

---

**Note**

The above rules mean that if a VPE doorbell is requested and a corresponding event is generated, an event is only generated once, until it is requested again, because the doorbell stops being requested when an event is generated.

---

$S_{RGZBY}$ Once a VPE doorbell event has been generated, no further VPE doorbell events will be generated for that VPE until the VPE doorbell is requested again. The VPE doorbell may be requested again by hypervisor software after making the VPE resident and subsequently making it non-resident.

$I_{BWQVY}$ A doorbell priority mask is set when the VPE makes a VPE non-resident and requests a VPE doorbell. The IRS Domain only generates a VPE doorbell if there is a virtual interrupt for the VPE with a priority greater than or equal to the minimum priority programmed in the doorbell priority mask, accessible via IRS_VPE_DBR.DBPM.

$S_{WWFWZ}$ When making a VPE non-resident, GICH_CONTEXTR_EL2.DBPM sets the minimum priority for a virtual interrupt to trigger the VPE doorbell. Software can use this field to ensure that doorbells are only generated for virtual interrupts which would be handled by the VPE if it were resident.

$R_{DXSYV}$ When the VPE doorbell conditions are met, the VPE doorbell interrupt event is generated in finite time.

If the VPE doorbell conditions are met for a short amount of time and are then no longer met, for example due to a reconfiguration of an interrupt, and the VPE doorbell event was not yet generated, it is IMPLEMENTATION DEFINED whether a VPE doorbell event is generated.

$R_{RFWPN}$ If a VPE doorbell event is generated for an INTID that is unreachable, the event is treated as any other interrupt event for an unreachable INTID and the event does not make any interrupt Pending.

$I_{XJQLG}$ If a VPE doorbell event is generated for an INTID that is unreachable, the doorbell is no longer requested for the VPE.

$I_{DTTDM}$ If a VPE doorbell event is generated for an INTID that is unreachable, and software error reporting is supported, this error is reported in IRS_SWERR_STATUSR.

$R_{DHHRM}$ If a VPE doorbell uses an LPI where any of the following are true, it is CONSTRAINED UNPREDICTABLE whether the IRS makes the LPI Pending when the VPE doorbell event is generated, and whether any IRS synchronization requests affect the LPI:

- There is an ITS mapping to the LPI.
- There is another VPE doorbell that uses the LPI.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

See also:

- 2.10.6 *Requesting VPE doorbells*

### 4.10.8 1ofN doorbells

GTHLBD The architecture supports generating a VPE doorbell event when there is a virtual 1ofN interrupt for a VM and there are no resident VPEs with 1ofN PE selection enabled for the VM. This allows a hypervisor to make scheduling decisions about when to run a VPE to handle a 1ofN interrupt.

DFJTLC A *1ofN doorbell* is the mechanism that generates a VPE doorbell when the 1ofN doorbell conditions are met. The VPE doorbell is generated for the *1ofN VPE doorbell target*.

IJRYKN The 1ofN doorbell configuration for a VM is managed by selecting the VM and accessing IRS_VM_DBR.

Software selects a VM by writing the VM ID to IRS_VM_SELR.VM_ID.

IRS_VM_STATUSR.IDLE reports whether the effects of the write IRS_VM_SELR are complete.

IRS_VM_STATUSR.V reports whether the value written to IRS_VM_SELR selects a valid VM.

IRS_VM_STATUSR.V is updated on a write to any of the following registers:

- IRS_VM_SELR.
- IRS_VM_DBR.

INWBYP Whether 1ofN doorbells are enabled or disabled for the VM is programmed in IRS_VM_DBR.EN.

IFKZMP The 1ofN VPE doorbell target is programmed in IRS_VM_DBR.VPE_ID.

DQTNVT The *1ofN doorbell conditions* are met when all of the following are true:

- Virtual 1ofN interrupt routing is supported.
- There is no VPE belonging to the VM where all of the following are true:
  - The VPE is resident.
  - The VPE has 1ofN PE selection enabled.
- 1ofN doorbells are enabled for the VM.
- There is at least one virtual 1ofN interrupt for the VM that is Pending, Inactive, and Enabled.

See 4.10.7 *VPE doorbells* for more information about additional conditions for generating a VPE doorbell event.

RVVLPV When a VM is made valid, 1ofN doorbells are disabled for the VM.

### 4.10.9 Save and restore of virtual interrupts

GNSYHS The architecture is designed to support saving the state and configuration of all virtual interrupts in a VM to memory, and restoring the saved state and configuration from memory.

This is to support common virtualization use cases such as VM live migration across physical hosts, snapshotting of VM state, and provisioning of VMs from a pre-booted state.

DCZQCL This chapter refers to saving the state of a *source* VM and restoring it to a *destination* VM.

This is regardless of whether the source and destination VMs are running on separate physical machines, are separate VMs with distinct VM IDs on the same physical machine, or are separate instances of a VM executing at different times with the same VM ID on the same physical machine.

DMGVXX A VM is *Quiesced* if all of the following are true:

- No VPEs in the VM are resident.
- There is no change in the Pending state of any virtual interrupt for the VM.
- There is no change to the number of valid PEs in the VM.
- The VM does not become invalid.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{GNLWX}$ To Quiesce a VM, software should ensure that all of the following are true:

- The Pending states of LPIs are not updated by events from an ITS.
- The Pending states of SPIs are not updated by events generated by the IRS.
- The Pending states of LPIs or SPIs are not updated as a result of executing the `GIC xDPEND` instruction on a PE.
- No VPE is resident.
- The virtual LPI IST and virtual SPI IST do not become valid or invalid.
- No VPE in the VM is made valid.
- The VM is not made invalid.

$S_{JVVJM}$ Software can prevent updates to virtual interrupts belonging to a VM when saving the state of virtual interrupts by using any of the following mechanisms:

- Preventing devices from generating interrupts for the VM.
- Unmapping ITS events that are mapped to virtual interrupts belonging to the VM.
- Issuing a synchronization request on every ITS that had events mapped to virtual interrupts belonging to the VM.
- Unassigning SPIs that are assigned to virtual SPIs in the VM.
- Issuing a synchronization request on all IRSs in the system.

$I_{ZKFGL}$ The state and configuration of virtual interrupts are written to the virtual ISTs by writing the VM ID to IRS_SAVE_VMR.VM_ID and writing 1 to IRS_SAVE_VMR.S when the VM is Quiesced.

The operation is complete when IRS_SAVE_VM_STATUSR.IDLE is 1.

Following a write that sets IRS_SAVE_VMR.S or IRS_SAVE_VMR.Q to 1, IRS_SAVE_VM_STATUSR.Q reports whether the VM is Quiesced since the last write that set IRS_SAVE_VMR.S to 1.

---

**Note**

Saving the state and configuration of virtual interrupts applies to the ISTs that are valid for the VM. For example, if a VM does not have a virtual SPI IST but does have a valid virtual LPI IST, writing 1 to IRS_SAVE_VMR.S saves the state of the virtual LPIs to the virtual LPI IST.

---

$R_{XPMBB}$ Following a write that sets IRS_SAVE_VMR.S to 1, for as long as the specified VM is Quiesced, the values stored in the virtual ISTs are consistent with the state and configuration of the virtual interrupts.

$S_{JQMTQ}$ The following software sequence can be used to save the state and configuration of virtual interrupts for a VM:

1. Quiesce the set of VM.
2. Write 1 to IRS_SAVE_VMR.S and poll IRS_SAVE_VM_STATUSR.IDLE until it returns 1 on the same IRS.
3. Check that the VM remained in the Quiesced state during the save operation by checking that IRS_SAVE_VM_STATUSR.Q returns 1. This ensures that the data is written correctly to the virtual IST.
4. Copy the virtual ISTs to a new location.
5. Write {0, 1} to IRS_SAVE_VMR.{S, Q} and poll IRS_SAVE_VM_STATUSR.IDLE until it returns 1 on the same IRS.
6. Check that the VM remained in the Quiesced state during the copy operation by checking that IRS_SAVE_VM_STATUSR.Q returns 1. This ensures that the virtual IST remained unmodified during the copy operation.

$I_{DBSGD}$ Saving the state and configuration of virtual interrupts can be requested on any IRS and applies to all virtual interrupts for the VM regardless of which IRS the request is made on. In a multi-IRS system, an implementation coordinates this operation across all IRSs.

$R_{CFDQJ}$ If a write occurs to IRS_SAVE_VMR when IRS_SAVE_VM_STATUSR.IDLE is 0 on any other IRS for the same IRS Domain, it is CONSTRAINED UNPREDICTABLE whether the state and configuration of virtual interrupts are saved to the virtual ISTs and the value returned in IRS_SAVE_VM_STATUSR.Q is UNKNOWN on both IRSs.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{SYNGG}$ Following a write that sets IRS_SAVE_VMR.S to 1, the state and configuration of virtual interrupts are stored in the virtual ISTs using the format defined in L2_ISTE.

$S_{JKLZK}$ To support saving the state of a source VM and restoring it to a destination VM, software is expected to perform the following steps:

1. Quiesce the source VM.
2. Save the state and configuration of all virtual interrupts to the virtual ISTs.
3. Save a copy of the virtual IST from the source VM.
4. Scan the copy of the virtual IST and record which virtual interrupts are Pending.
5. Update all entries in the copy of the virtual IST as follows:
    1. Set the L2_ISTE.Pending field to 0.
    2. Set IAFFID to 0 if IRM is 1 and IRS_IDR0.ONE_N is 1.
    3. Set the metadata area to 0 if IRS_IDR2.ISTMD is 1.
6. Configure the destination VM to use the copy of the virtual IST as the new IST.
7. Make the new virtual IST valid.
8. For each virtual interrupt that was recorded as Pending, make the virtual interrupt Pending using the `GIC VDPEND` System instruction.

The destination VM must be configured with valid VPEs before making the virtual interrupts Pending to ensure that the interrupts are considered for the virtual candidate PPI for the VPEs.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.11 IRS power management

$G_{CRTDH}$ The architecture supports implementations that enable the following power management operations:

- System suspend and resume: All interrupt state and configuration, and register state of all IRSs in the system for all Interrupt Domains, are stored to memory on suspend and restored from memory on resume.
- Opportunistic power management: The system may decide to power down one or more IRSs while preserving the illusion to software that the IRSs are fully operational.

$R_{FYSXL}$ Support for system suspend and resume is supported using an IMPLEMENTATION DEFINED sequence.

The IMPLEMENTATION DEFINED sequence must support saving all of the following to memory for all Interrupt Domains:

- Physical LPI state and configuration.
- Physical SPI state and configuration.
- If virtualization is supported in the Interrupt Domain, GIC-managed state related to the VM table, including all of the following:
  - State and configuration of virtual interrupts stored in the virtual LPIs.
  - State stored in the VM descriptors, including the configuration of VMs.
  - State stored in the VPE descriptors, including the configuration of VPEs.

The IMPLEMENTATION DEFINED sequence must support restoring all of the information from memory to the values saved during suspend.

$I_{XHGLK}$ Data structures used by mechanisms for IRS save and restore for system suspend are IMPLEMENTATION DEFINED.

$S_{NWXQM}$ Arm expects that system suspend and resume is managed by system-specific software.

$R_{HJFWL}$ Support for opportunistic power management is IMPLEMENTATION DEFINED. The opportunistic power management sequence must preserve the illusion to software that the IRSs remain powered on.

$D_{BGYZW}$ Each PE connected to an IRS is either *offline* or *online*.

Arm expects that a PE is offline when the PE is physically powered off and is unable to execute GIC System instructions or write to GIC System registers.

$I_{TCRZH}$ Whether a PE is online or offline is reported in IRS_PE_STATUSR.ONLINE for the PE selected in IRS_PE_SELR.

$R_{PMSHG}$ The mechanism to determine whether a PE is offline or online is IMPLEMENTATION DEFINED.

$D_{XGKCK}$ When a PE is offline, the IRS generates a *Wake Request* to a PE to bring it online.

$R_{CLDWD}$ The mechanism through which an IRS generates a Wake Request to a PE is IMPLEMENTATION DEFINED.

$R_{HPCRY}$ When the IRS generates a Wake Request to a PE, the request is generated in finite time.

The Wake Request does not guarantee that the PE becomes online. The Wake Request is treated as a hint to the power control subsystem.

$I_{FYYJF}$ When a PE becomes offline, and the PE is not expected to become online when receiving a Wake Request, Arm strongly recommends that the system is configured to avoid sending Wake Requests to the PE.

The system is configured not to generate Wake Requests to the PE when all of the following are true for all IRS Domains:

- No Targeted interrupts are programmed to target the PE.
- 1ofN PE selection is disabled for the PE.

An example of when a PE is not expected to become online when receiving a Wake Request is PSCI_CPU_OFF. See *Arm® Power State Coordination Interface*[7] for more information about PSCI.

$R_{YPBHL}$ When a PE is offline, all of the following are true:

- The PE does not execute any instructions that generate Interrupt Effects.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- If the IRS has selected a physical candidate HPPI for a PE for any Interrupt Domain, and the IRS is enabled for that Interrupt Domain, the IRS generates a Wake Request to the PE.

When a PE is online, the IRS does not generate any Wake Requests to the PE.

$R_{FVFKC}$ When a PE becomes offline and the IRS has selected any candidate HPPIs for the PE, each candidate HPPI is either acknowledged before the PE becomes offline or is not acknowledged while the PE is offline.

When the PE subsequently becomes online, any selected candidate HPPIs for the PE for enabled Interrupt Domains, can be acknoweldged by the PE.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.12 IRS memory access rules

$R_{VHRSJ}$ All IRS data structures are little-endian.

$R_{FRXXQ}$ The IRS accesses all its data structures using Normal memory types. See [1] for more information about memory access types.

$S_{XFKQH}$ Software allocates memory for the data structures and programs the physical address of the allocated data structures in IRS registers. Arm expects that such allocations will typically be done from Conventional memory.

$I_{DPJZK}$ IRS_CR1 specifies the Shareability and Cacheability attributes for memory accesses performed by the IRS.

Changes to IRS_CR1 can only be made when the IRS does not access memory, specifically when no data structures are valid on the IRS.

The most recent values written to IRS_CR1 are used when the IRS accesses memory following a write to make any data structure valid.

Arm recommends that IRS_CR1 is initialized with the same Shareability and Cacheability attributes on all IRSs in the system for an IRS Domain before making any data structure valid.

$R_{XKMVR}$ An IRS does not access any memory location which is not derived from address and configuration data stored in registers and structures belonging to the IRS Domain.

When a memory address is stored in an address field in a register or data structure entry, and that register or entry includes a field indicating that the address is not valid, the IRS does not derive any memory location from that address field.

This includes scenarios in which the IRS behaviour is CONSTRAINED UNPREDICTABLE, UNPREDICTABLE, or IMPLEMENTATION DEFINED.

$I_{GTTMT}$ When a memory address is stored in a data structure, and that entry includes a field indicating whether the address is valid, the address only becomes invalid as a result of a write to an IRS map register.

In this case, IRS Domain is guaranteed to not derive an address from the entry once the effects of the write to the map register are complete.

$R_{CPXND}$ When the IRS accesses an IRS data structure, and the contents of that data structure are not defined as UNKNOWN at the time of the access, the access must be a 64-bit aligned, single-copy atomic access of at least 64 bits in size.

For data structures where the entries are smaller than 64 bits, accesses are permitted to be smaller than 64 bits, provided that each access is:

- Single-copy atomic,
- equal to the size of the entry, and
- aligned to the size of the entry.

$R_{LRRFV}$ If an IRS access to any IRS data structure is performed into any PCIe address space, then the IRS is permitted to return an UNKNOWN value or terminate the access and report the error.

In this case, the IRS is permitted to stop the operation as follows:

- If the operation was due to an incoming interrupt event, the IRS is permitted to drop the interrupt event.
- If the operation was requested as the result of executing a GIC System instruction or writing to a GIC System register, the IRS is permitted to indicate a failure back to the PE or IGNORE the operation.

$R_{RFYLF}$ When an IRS accesses a memory location in a PAS, it relies on information stored in registers and data structures structures that are accessible only within the same PAS to validate that the access is permitted.

This includes scenarios in which the IRS behavior is CONSTRAINED UNPREDICTABLE, UNPREDICTABLE, or IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

For example, an IRS access to a data structure is permitted only if the memory location lies within a region defined by a base address and size held in a register or another data structure entry, where that defining information resides in the same PAS as the access itself.

This prevents software running in a Security state from directing the IRS to access any memory location using another PAS than the one associated with the software's Security state.

$I_{FJRHT}$ If the IRS experiences an external abort during a memory access to an IRS data structure, the IRS stops the operation.

If software error reporting is supported, the error is reported with IRS_SWERR_STATUSR.EC.

$R_{LVNHK}$ If the IRS experiences an external abort during a memory access to an IRS data structure, it is IMPLEMENTATION DEFINED whether the IRS reports a RAS error using an IMPLEMENTATION DEFINED mechanism.

$R_{GZJNW}$ If the IRS experiences an external abort as part of an operation that occurs as a result of a GIC System instruction or write to a GIC System register on a PE, the effects of the instruction are UNKNOWN and the IRS may optionally report an error.

$I_{LQCPD}$ For example, if the IRS experiences an external abort when trying to access the IST as part of handling a `GIC xDDIS` instruction, the instruction is permitted to execute as a NOP and the IRS is permitted to report an error using the software error reporting mechanism and by raising IMPLEMENTATION DEFINED RAS errors if the abort is caused by a memory system error.

$I_{DPBMS}$ As another example, if the IRS experiences an external abort when trying to access the IST as part of handling a `GIC xDRCFG` instruction, the IRS is permitted to indicate failure to the PE that results in ICC_ICSR_EL1.F being set to 1, and the IRS may optionally report an error.

$R_{KMZHR}$ If the IRS experiences an external abort as part of processing an interrupt that is being signaled, the effects on the state and configuration of the signaled interrupt is UNKNOWN and the IRS is permitted to drop the event. The IRS may optionally report an error using the software error reporting mechanism and by raising IMPLEMENTATION DEFINED RAS errors if the abort is caused by a memory system error.

$R_{ZRWQP}$ If an IRS data structure overlaps with any other IRS or ITS data structure, the IRS behavior is UNPREDICTABLE. This applies to both IRS data structures for physical and virtual interrupts.

The UNPREDICTABLE behavior may result in loss of interrupt configuration and state, but must not result in access to memory outside the PAS associated with the IRS Domain or in access to any memory location which is not derived from address and configuration data in valid IRS registers.

Software can recover from this situation by performing the following sequence:

1. Disabling the ITS.
2. Making the corresponding data structures invalid in the IRS.
3. Reconfigure the base addresses of the data structure to avoid any overlap.

See also 5.3 *Translation structures*.

$I_{QTYKR}$ The following changes to system state when an interrupt event is processed by an IRS are not required to result in a write to memory:

- Interrupt Write effects generated by interrupt events.
- Interrupt Write effects generated by GIC System instructions executed on a PE.
- Configuration changes generated by GIC System instructions executed on a PE.
- Configuration changes generated by writes to IRS registers.

In these cases, the change to system state does not correspond directly to any Memory Write effect to IRS data structures.

However, the change may cause an update to architecturally invisible IRS caches and Memory Write effects to write back cached data to IRS data structures where the content of the data structure is described as UNKNOWN.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{FZBVM}$ An IRS is permitted to implement caching of interrupt state, VMs, and VPE configurations using architecturally invisible IRS-specific caches.

The IRS caches are not considered data or system caches belonging to the memory system, meaning they are not managed by any cache maintenance instructions issued by a PE.

When an IRS data structure is invalid, the IRS does not cache previously accessed data in IRS-specific caches.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.13 IRS support for MPAM

$I_{LKQJP}$ The Memory System Resource Partitioning and Monitoring Memory, MPAM, architecture defines per-transaction attributes that affect system behavior or the behavior of components that the transactions pass through or Completers that satisfy a transaction [1].

The additional attributes are two identifiers:

- Partition ID, or PARTID.
- Performance Monitoring Group, or PMG.

The PARTID and PMG are both interpreted within a PARTID space. The PARTID space used depends on the Security state associated with an IRS Domain, and the PARTID and PMG values used may be programmed independently for each IRS Domain. See [1] for more information about MPAM information bundles.

$I_{WXHXM}$ In the IRS, support for MPAM, is indicated in IRS_IDR0.MPAM.

If MPAM is supported, the supported PARTID and PMG width is indicated in IRS_MPAM_IDR.{PARTID_MAX,PMG_MAX}.

$R_{RMZLS}$ If MPAM is supported, IRS accesses to memory are associated with the PARTID and PMG programmed in IRS_MPAM_PARTID_R.{PARTID,PMG}.

$I_{NSQVT}$ In systems without support for RME, the PARTID space used by the IRS is determined by the memory system attribute MPAM_NS.

In systems with support for RME, the PARTID space used by the IRS is determined by the memory system attribute MPAM_SP.

$I_{HHRJR}$ The IRS architecture has optional support for MPAM PARTID space selection indicated by IRS_MPAM_IDR.HAS_MPAM_SP.

If IRS_MPAM_IDR.HAS_MPAM_SP is 0, each IRS for each Interrupt Domain uses a default MPAM PARTID space.

The following table shows the MPAM PARTID space used for accesses made by the IRSs for each Interrupt Domain if IRS_MPAM_IDR.HAS_MPAM_SP is 0 and the system does not support RME:

| Interrupt Domain | MPAM PARTID space |
|---|---|
| Secure | Secure PARTID space |
| Non-Secure | Non-secure PARTID space |
| EL3 | Secure PARTID space |

The following table shows the MPAM PARTID space used for accesses made by the IRSs for each Interrupt Domain if IRS_MPAM_IDR.HAS_MPAM_SP is 0 and the system supports RME:

| Interrupt Domain | MPAM PARTID space |
|---|---|
| Secure | Secure PARTID space |
| Non-Secure | Non-secure PARTID space |
| EL3 | Root PARTID space. |
| Realm | Realm PARTID space. |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

If IRS_MPAM_IDR.HAS_MPAM_SP is 1, the IRS uses the MPAM PARTID specified by IRS_MPAM_PARTID_R.MPAM_SP.

$R_{NWLJQ}$ If an IRS without support for MPAM is integrated in a system that supports MPAM, the PARTID and PMG used for each IRS for each Interrupt Domain is IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.14 IRS support for Memory Encryption Contexts

$I_{FPYMT}$ The Memory Encryption Contexts feature, FEAT_MEC, provides finer-grained memory encryption contexts, within the Realm physical address space, to be assigned to Realms, with policy controlled by Realm EL2 [1].

$G_{HJFYY}$ The GICv5 architecture ensures that an IRS can function correctly in systems with support for Memory Encryption Contexts (MEC).

$I_{HJNXC}$ In a system with support for MEC, data may be shared between the PEs and an IRS in the following scenarios:

- A PE allocates and initializes data structures used by an IRS.
- An IRS writes the contents of virtual ISTs to memory to support VM migration.

In such a system, all IRS Realm data structures are managed and configured by Realm EL2, and virtualized software running at EL1 is not expected to have direct access to the IRS configuration registers. Therefore, the IRS only supports configuration of a single MECID used for all IRS memory accesses in the Realm PAS. This allows software running at Realm EL2 to initialize data structures that are read by the IRS and allows the IRS to write the content of the virtual ISTs that are read by software running at Realm EL2.

$I_{YQFXX}$ Support for Memory Encryption Contexts (MEC) for an IRS is indicated in IRS_IDR0.MEC for the Realm Interrupt Domain. If the MEC feature is supported, the supported MECID width is indicated in IRS_MEC_IDR.MECIDSIZE for the Realm Interrupt Domain.

Arm strongly recommends that the MECID bit width supported by the IRS matches or exceeds the width supported by the PEs in the system.

$R_{YGVPR}$ If the MEC feature is supported, IRS accesses to memory are associated with a MECID that identifies the Memory Encryption Context of the access.

$R_{QKQQW}$ Accesses made by the IRS for Secure, Non-secure, and Root PA spaces are issued with the default MECID of zero.

$I_{TKDGT}$ Accesses made by the IRS to Realm PA space are associated with the global Realm PAS IRS MECID programmed in IRS_MEC_MECID_R.MECID.

$R_{KCCQN}$ If an IRS without support for the Realm Interrupt Domain is integrated in a system that supports MEC, all IRS accesses for that IRS are treated as having the default MECID of zero.

$R_{GMPJJ}$ In a multi-IRS system, if all of the following are true, the IRS Domain behavior is UNPREDICTABLE:

- One of the following is true:
  - IRS_SET_BASE_CFGR.VALID is 1.
  - IRS_CLR_MT_BASE_CFGR.VALID is 1.
- An IRS has a different MECID than another IRS, for the same PA space.

The UNPREDICTABLE behavior must not result in access to memory outside the PAS associated with the Interrupt Domain or an access to memory that does not follow the behaviors described in 4.12 *IRS memory access rules*.

To recover from the UNPREDICTABLE behavior, the IRS data structures are made invalid and the same MECID is configured across all IRSs for the same PA space.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 4.15 IRS support for software error reporting

$I_{GZRSS}$ The IRS specifies a mechanism to report errors because of incorrect programming.

$R_{QWVQB}$ The IRS detects software errors when it performs any of the following operations:

- The IRS processes an interrupt event in response to a signaled interrupt.
- The IRS accesses state and configuration of interrupts in response to any of the following events:
  - A System register access in the PE interface.
  - Execution of a GIC system instruction in the PE interface.
  - An MMIO access via the IRS configuration register frame.

$I_{TQQTS}$ Updates to the state and configuration of interrupts via the PE interface or the MMIO interface can be cached in the IRS caches. The IRS could experience an external abort, when it commits the cached information to the IRS data structures in memory at a specified address. The IRS uses the software error reporting mechanism to indicate such errors.

For example, an update to the configuration of a physical LPI is cached in the IRS caches. The address of the physical IST in IRS_IST_BASER.ADDR is invalid, and the IRS experiences an external abort when it commits the cached information to the L2_ISTE in memory. The IRS sets IRS_SWERR_STATUSR.EC to report this error.

Due to the effects of caching, this error could be reported after the operation to update the configuration of the physical LPI has completed from software's perspective. This means that two reads of IRS_SWERR_STATUSR can report errors without any interrupts or GIC System Instructions executed on a PE having been processed by the IRS between the two reads.

$R_{YHKJZ}$ IRS support for software error reporting is optional.

$I_{NMHQJ}$ IRS_IDR0.SWE reports whether software error reporting is supported.

$R_{HWQTK}$ The value of IRS_IDR0.SWE is the same for all Interrupt Domains in an IRS in the system.

$I_{ZTBKN}$ When IRS_IDR0.SWE is 1, the IRS uses the IRS_SWERR_STATUSR.IMP_EC to report any IMPLEMENTATION DEFINED errors detected by the IRS.

$I_{NLLSH}$ IRS_SWERR_STATUSR.EC is 0 when an IMPLEMENTATION DEFINED error is reported by the IRS.

$I_{YMQZW}$ Software error information can be read on each IRS for each Interrupt Domain via the following registers:

- IRS_SWERR_STATUSR.
- IRS_SWERR_SYNDROMER0.
- IRS_SWERR_SYNDROMER1.

$I_{FCYCT}$ When a software error is reported, the value of IRS_SWERR_STATUSR.V is 1. Otherwise, no software error is reported and fields in this register are UNKNOWN.

$R_{SNJBM}$ When software error reporting is implemented, the IRS reports a detected error using any of the defined error codes or sets IRS_SWERR_STATUSR.OF.

The IRS is not permitted to silently ignore an error that could be reported.

$I_{QKKNX}$ When more than one error code can be used to describe a detected error, it is IMPLEMENTATION DEFINED which error code is used.

The architecture does not define a priority between error codes that can occur at the same time.

$I_{ZLMMJ}$ When the value of IRS_SWERR_STATUSR.V is 1, all of the following are true:

- IRS_SWERR_STATUSR.EC specifies the fault that caused the software error.
- IRS_SWERR_STATUSR.S0V indicates whether IRS_SWERR_SYNDROMER0 contains valid error syndrome information.
- IRS_SWERR_STATUSR.S1V indicates whether IRS_SWERR_SYNDROMER1 contains valid error syndrome information.
- IRS_SWERR_STATUSR.OF indicates whether multiple software errors were detected.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{DMCGM}$ When IRS_SWERR_STATUSR.S0V is 1, IRS_SWERR_SYNDROMER0 reports the following information for the software error:

- The type and ID of the interrupt that resulted in the software error.
- Whether the interrupt is virtual or physical.
- If the interrupt is virtual, the VM ID of the VM that the interrupt is assigned to.

$I_{HQSCK}$ When IRS_SWERR_STATUSR.S1V is 1, IRS_SWERR_SYNDROMER1 reports the address of the IRS data structure associated with the software error.

$R_{XBYRF}$ For each reported error, the values of IRS_SWERR_STATUSR.{S0V,S1V} are IMPLEMENTATION DEFINED and set independently.

$S_{MTHCH}$ Arm recommends that software performs the following sequence to either clear the last error, or detect whether new errors were reported:

1. Read IRS_SWERR_STATUSR and determine which fields need to be cleared to zero.
2. In a single-copy atomic write to IRS_SWERR_STATUSR:
    1. Write ones to all the W1C fields that are nonzero in the read value.
    2. Write zero to all the W1C fields that are zero in the read value.
    3. Write zero to all the RW fields.
3. Read back IRS_SWERR_STATUSR after the write. If the value read back is the same as the value that was written, all W1C fields that were non-zero are cleared and no new errors were reported. Otherwise, one or more new errors were reported.

See also:

- IRS_SWERR_STATUSR.
- IRS_SWERR_SYNDROMER0.
- IRS_SWERR_SYNDROMER1.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 5
# Interrupt translation service (ITS)

RETIRED

$D_{MBMKY}$ The Interrupt Translation Service (ITS) is responsible for generating *ITS events* and translating them into *interrupt events* for delivery to the associated Interrupt Routing Service (IRS) in each Interrupt Domain.

Each translation specifies whether the resulting interrupt is physical or virtual, the associated Interrupt Domain, the interrupt's INTID and, if the interrupt is virtual, the VMID of the target VM.

The ITS uses translation structures in the form of a Device Table (DT) and Interrupt Translation Tables (ITT). These translation structures reside in memory and may be partially cached by the ITS. For more information, see 5.3 *Translation structures*.

Figure 5.1 illustrates the overall ITS translation flow.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

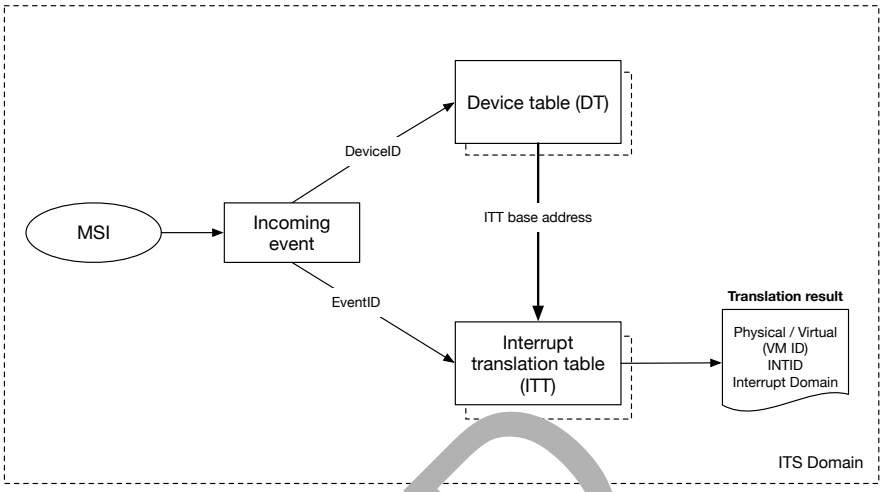



**Figure 5.1: GIC v5 ITS translation flow**

IVHDBC Translation of an ITS event may involve internal cache lookups and memory accesses to retrieve translation data.

IQQNWW An example of the ITS translating an event into an interrupt occurs when a Message Signaled Interrupt (MSI) is generated by a device. In this case, the MSI write targets an ITS translation register. The ITS captures the write and generates an ITS event. This event contains the DeviceID of the originator of the write and the EventID value provided in the write data. The ITS uses the programmed entries in the Device Table (DT) and Interrupt Translation Table (ITT) to translate the ITS event into an interrupt. The resulting interrupt is then forwarded to the associated IRS for delivery to a PE.

RZZQCC An ITS event is generated by the ITS when any of the following occurs:

- A memory write to ITS_TRANSLATER or ITS_RL_TRANSLATER.
- A write to ITS_GEN_EVENTR.
- An IWB forwards an event using an IMPLEMENTATION DEFINED communication mechanism.
- A system peripheral forwards an event using an IMPLEMENTATION DEFINED communication mechanism.

DXCCZK An ITS event contains all the following information:

- DeviceID.
- EventID.
- Interrupt event type.
- Originating Interrupt Domain.

DZVRSC The *originating Interrupt Domain* of an ITS event is determined as follows:

- If the event is generated by a write to ITS_TRANSLATER, ITS_RL_TRANSLATER, or ITS_GEN_EVENTR, the originating Interrupt Domain is the ITS Domain associated with the PAS of the register.
- If the event is generated by an IWB, the originating Interrupt Domain is the Interrupt Domain that the corresponding wire is assigned to.
- If the event is generated by a system peripheral using an IMPLEMENTATION DEFINED mechanism, the originating Interrupt Domain is determined using an IMPLEMENTATION DEFINED mechanism.

**Note**

The originating Interrupt Domain may not correspond to the ITS Domain that translates the event. See 5.1 *ITS Domains* for more information.

RRTVBF When multiple SET_LEVEL and CLEAR events are received for the same DeviceID and EventID, they must be translated and forwarded to the IRS in the order in which they are received.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{BZTJM}$ When the ITS receives a memory write transaction to ITS_TRANSLATER, the ITS generates an ITS event that is translated by the ITS in finite time.

$I_{NBTNK}$ ITS events can be Message Signaled Interrupts (MSIs), for example generated by PCIe endpoints, or can be events generated by IWBs or other system components.

Arm expects that MSIs are typically generated as a result of an MSI source performing a write to ITS_TRANSLATER.

$S_{HFCND}$ Software ensures that a PCIe posted write to an ITS_TRANSLATER_FRAME is Accepted by the ITS, by ensuring that the PCIe subsystem issues a Completion TLP following the posted MSI write.

See *PCI Express® Base Specification Revision 6.0*[8] for more information on the PCIe TLP ordering rules.

$R_{CHRYR}$ For an incoming memory write transaction to ITS_TRANSLATER Accepted by the ITS, the ITS handles the incoming write independently from the state of the source.

$I_{CNGJZ}$ The ITS Accepts an incoming memory write transaction to ITS_TRANSLATER without waiting for the IRS to Accept a corresponding interrupt event and without waiting for the IRS to present the corresponding interrupt to a PE.

The IRS separates the process of making an interrupt Pending from signaling the interrupt to the PE.

See Chapter 4 *Interrupt routing service (IRS)* for more information.

$I_{YDBCQ}$ Interrupt events sent by an ITS to the associated IRS contain all of the following:

- LPI ID.
- Physical or virtual interrupt.
- VM ID if virtual.
- Interrupt event type.
- Interrupt Domain.

$R_{NYJRF}$ An Interrupt event is never sent to the associated IRS speculatively. It is only sent when an ITS event is generated and translated by the ITS.

$I_{PTZXC}$ The mechanism used by the ITS to communicate events to the IRS is IMPLEMENTATION DEFINED. See 3.2 *Communication between GIC system components* for more information.

$R_{HPDKY}$ An ITS is associated with a single IRS and all interrupt events generated by an ITS are sent to that IRS.

$R_{DXDFC}$ The DeviceID in an ITS event is a unique identifier within the DeviceID namespace of each ITS. The DeviceID is unique for each device that can generate events to the ITS.

For example, Arm expects that the 16-bit Requester ID from a PCIe Root Complex is used to derive the DeviceID.

**Note**

Each ITS has a separate namespace for the DeviceID which is shared across the Interrupt Domains.

$R_{DJZVX}$ The number of DeviceID bits supported by an ITS is in the range from 0 to 32 inclusively, and is reported by ITS_IDR1.DEVICEID_BITS.

$I_{VCJFT}$ An ITS can support 0 DeviceID bits, which means that the ITS only supports a single device.

$R_{WXGDP}$ The DeviceID namespace is unique for each ITS.

For example, DeviceID 0 on ITS A identifies a device distinct from the device identified by DeviceID 0 on ITS B.

$R_{CMZTW}$ The number of EventID bits supported by an ITS is in the range from 0 to 16 inclusively, and is reported by ITS_IDR2.EVENTID_BITS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{WCDFX}$ The EventID namespace is separate for each ITS DeviceID.

For example, EventID 0 for DeviceID 0 on ITS A identifies an event distinct from the event identified by EventID 0 for DeviceID 1 on ITS A.

$I_{TNTVS}$ The number of EventID bits supported for a DeviceID is programmed in L2_DTE.EVENTID_BITS. See 5.3.2 *The Interrupt Translation Table (ITT)* for more information.

$R_{RYJGM}$ The mechanism by which a DeviceID is communicated to the ITS is IMPLEMENTATION DEFINED.

However, a unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS when a write to the ITS translation registers occurs in a manner that cannot be spoofed by any agent capable of performing writes.

$G_{TTLHL}$ The ITS translation mechanism supports *interrupt isolation*.

This allows an untrusted software agent such as a virtual machine to directly control the EventID used by an interrupt source without being able to generate interrupts with INTIDs used for other interrupt sources.

$I_{RWJXV}$ Interrupt isolation can be achieved through the use of an ITS by only allowing the hypervisor to control the ITS. The hypervisor therefore controls the translation of each DeviceID and EventID value pair into an INTID for the Physical Interrupt Domain or a specified VM. In this scenario, the ITS only translates EventIDs programmed by the hypervisor for a given DeviceID, and since the DeviceID is not under control of the VM, the hypervisor fully controls which INTIDs can be signaled irrespective of the EventID programmed by the VM.

$R_{QWVTC}$ For the interrupt isolation mechanism to work, the DeviceID used to identify a device context assigned to untrusted software must be static and not under direct or indirect control of the untrusted software.

$I_{WRXRW}$ Each ITS has a unique ITSID which is reported in GITS_IIDR.ITSID. The ITSID is unique for each ITS in the system.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 5.1 ITS Domains

$G_{QYXKS}$ An ITS operates independently for each Interrupt Domain.

$D_{WNGLR}$ The GICv5 architecture defines an *ITS Domain* which provides translation services for an Interrupt Domain. An *ITS Domain* comprises register state and translation structures independently configured from other ITS Domains on the same ITS.

An ITS Domain is associated with an Interrupt Domain and its corresponding PAS.

$I_{PXYHF}$ Each ITS Domain implements a separate ITS_CONFIG_FRAME. The ITSID reported in ITS_IDR0.ITSID is the same for each ITS_CONFIG_FRAME that belongs to the same ITS.

$R_{PMQQB}$ The ITS Domain that translates an ITS event is determined as follows:

- If the ITS event is generated by a write to ITS_TRANSLATER, the ITS Domain is the originating Interrupt Domain.
- If the ITS event is generated by a write to ITS_GEN_EVENTR, the ITS Domain is the ITS Domain specified by ITS_GEN_EVENTR.TARGET_DOMAIN.
- If the ITS event is generated by a write to ITS_RL_TRANSLATER, the ITS Domain is the Realm ITS Domain.
- If the ITS event is generated by an IWB, the ITS Domain is the originating Interrupt Domain.
- If the event is generated by a system peripheral using an IMPLEMENTATION DEFINED mechanism, the ITS Domain is the ITS Domain that corresponds to the originating Interrupt Domain.

$R_{YJNYY}$ If an ITS is associated with an IWB, the ITS provides an ITS Domain for each Interrupt Domain implemented by the IWB.

$R_{DZYMW}$ An ITS does not support any Interrupt Domain not supported by the IRSs.

$I_{XCBRD}$ The IRS associated with the ITS implements support for all the ITS Domains provided by the ITS, because an IRS implements support for all the Interrupt Domains supported by the PEs in the system.

$I_{KFVMT}$ An ITS exposes separate control register frames for each ITS Domain.

$I_{KKYFF}$ An ITS exposes separate translation register frames for each ITS Domain.

$I_{NJHMG}$ An ITS implements at least one ITS translate register frame, ITS_TRANSLATE_FRAME, for each ITS Domain. If an ITS Domain implements more than one translate register frame, the DeviceID space is shared across all the translate register frames belonging to the same ITS.

$R_{FGHHH}$ Each ITS Domain can have a maximum of 256 ITS translate register frames.

$S_{HPMNL}$ Arm expects that the ITS translation register frame identifier, as well as the relationship between ITS translate register frames and an ITS Domain and its configuration register frame, is communicated to system software via firmware data structures.

$I_{TNYVZ}$ The ITS does not provide a mechanism to identify via registers which ITS translation register frames are associated with which ITS configuration register frames.

$I_{GCJFM}$ Different from GICv3, the GICv5 architecture does not require the control register frame and translation register frame to be contiguous with respect to each other.

$I_{YCFZC}$ The base address of each register frame for each ITS Domain should be provided to software by firmware.

$I_{XFTTF}$ The ITS Domain for a register frame is reported in ITS_IDR0.INT_DOM.

$R_{PSXLG}$ All memory accesses performed by the ITS are subject to GPT checks.

An access is performed using the PAS associated with the ITS Domain where the translation is requested.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

IDRTDZ Figure 5.2 shows an overview of the ITS Domain registers and translation structures.

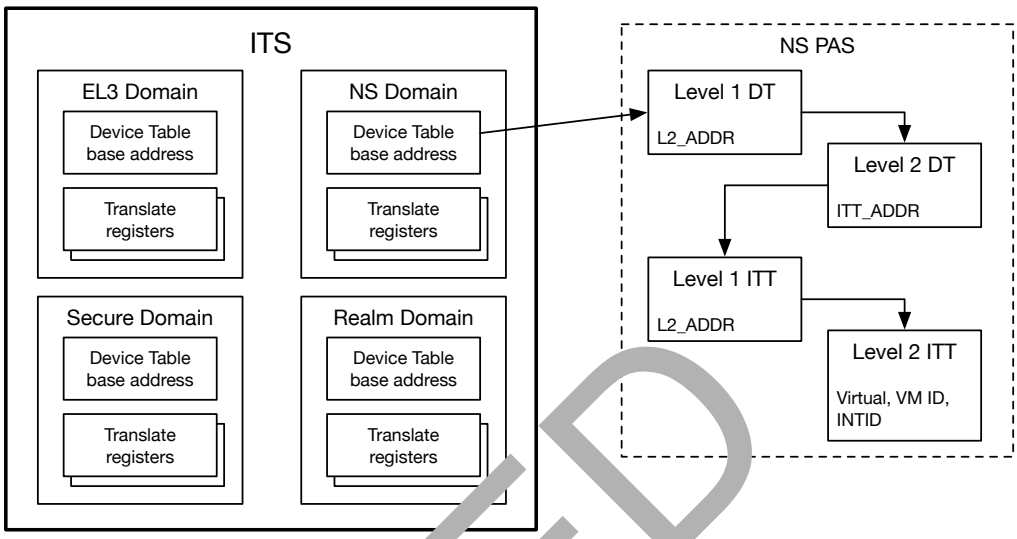



**Figure 5.2: GICv5 ITS Domains**

Translation structures are only shown for the Non-secure domain but exist separately for each ITS Domain.

ISRZCQ Arm expects an interrupt source to be configured to use the translation register frame of an ITS Domain corresponding to the Security state and PAS that manages the device. This allows end-to-end isolation of interrupt events and interrupt configuration within a security state. For example, a device managed by Software in the Secure state can generate MSIs to the Secure ITS Domain in the Secure PAS, and only software with access to the Secure PAS can configure the interrupt, and only that specific device can generate a Secure interrupt with the configured INTID.

RSXLJQ The following ITS Domains support virtual interrupts:

- Non-secure.
- Realm.
- Secure.

The EL3 ITS Domain does not support virtual interrupts.

RNTTVJ An event can be translated to generate interrupt events for a VM in the Interrupt Domain corresponding to the ITS Domain. This is a consequence of the VM ID namespace being separate in each Interrupt Domain.

See also:

- 3.1 *Interrupt Domains*

### 5.1.1 Supporting Realm interrupts from Non-secure writes

GYLHMP The GICv5 architecture supports direct injection of MSIs generated by Non-secure devices to Realms.

IBGWGH An ITS event is translated in the Realm ITS Domain, even though the event is generated as a result of a write to a register in the Non-secure PAS, if any of the following are true:

- The event is generated in response to a write to ITS_RL_TRANSLATER.
- The event is generated in response to a write to ITS_GEN_EVENTR in the Non-secure PAS, and the write sets ITS_GEN_EVENTR.TARGET_DOMAIN to `0b01`.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{JKXQV}$ Software in the Realm Security state controls whether an ITS event generated by a write to an ITS register in the Non-secure PAS is permitted to be translated in the Realm ITS Domain by programming L2_ITTE.DAC to allow such a translation.

See 5.3.2 *The Interrupt Translation Table (ITT)* for more information about the DAC field.

$S_{GYSSD}$ When an untrusted PCIe function is assigned to a VM, its MSI-X vectors are programmed to generate MSI writes to ITS_TRANSLATER. The MSI writes use the Non-secure PAS.

The ITS events are generated as a result of the write to a register in the Non-secure Interrupt Domain and translated in the Non-secure ITS Domain. If the translation is successful, LPIs are generated in the Non-secure Interrupt Domain.

When the function is assigned to Realm, its MSI-X vectors can be programmed to generate MSI writes to the ITS_RL_TRANSLATER. The MSI writes still use the Non-secure PAS, however, they are translated in the Realm ITS Domain. If the translation is successful, LPIs are generated in the Realm Interrupt Domain.

$S_{SZVBR}$ On a system that implements the Realm Management Extension, for an untrusted device that is assigned to Realm, there are scenarios where the host hypervisor must be able to inject a virtual LPI into Realm. The host hypervisor and the target VPE of Realm could be running on different PEs.

For example, an untrusted VirtIO[9] device is assigned to Realm. The device is emulated in the host hypervisor. The host hypervisor must be able to inject virtual LPIs into Realm.

The host hypervisor can write to ITS_GEN_EVENT with TARGET_DOMAIN set to `0b01` in the Non-secure PAS to generate these ITS events translated by the Realm ITS Domain.

See also:

- ITS_RL_TRANSLATER.
- L2_ITTE, Level 2 interrupt translation table entry.
- 5.3.2 *The Interrupt Translation Table (ITT)*.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 5.2 Operation

$R_{SXWCM}$ When ITS_CR0.ITSEN is 1, the ITS Domain is *enabled* and all of the following are true:

- The ITS translates ITS events and may generate corresponding interrupt messages to the associated IRS.
- The ITS Domain may perform memory accesses to the ITS Domain's translation structures as part of processing ITS events or as a result of handling ITS register accesses.

When ITS_CR0.ITSEN is 0 and ITS_CR0.IDLE is 1, the ITS Domain is *disabled* and all of the following are true:

- The ITS does not process ITS events; ITS events are dropped and have no effects on the ITS.
- The ITS Domain does not generate outgoing interrupt events to the associated IRS.
- The ITS Domain does not perform memory accesses to the ITS Domain's translation structures.

$R_{RNQHC}$ An ITS event is ignored if it cannot be translated by the ITS.

$I_{XFFRW}$ If software error reporting is supported, the ITS reports if an ITS event cannot be translated using the appropriate error code in ITS_SWERR_STATUSR.EC.

$I_{HYKFL}$ A change to ITS_CR0.ITSEN is not guaranteed to be observed by the ITS until ITS_CR0.IDLE is 1.

See also:

- 5.2.1 *Enabling and disabling the ITS*

### 5.2.1 Enabling and disabling the ITS

$R_{FXDRJ}$ When a write to ITS_CR0.ITSEN changes the value from 0 to 1, the ITS Domain begins a transition from disabled to enabled, and all of the following are true:

- The transition completes in finite time.
- The transition is complete when ITS_CR0.IDLE is 1.
- The ITS will process all events that are Accepted after the transition is complete.

---

**Note**

While the ITS is transitioning from disabled to enabled, transactions arriving at the ITS will observe the ITS being enabled or disabled. The architecture only guarantees transactions arriving after the transition from disabled to enabled is complete to observe that the ITS is enabled.

---

$R_{YSGFF}$ When a write to ITS_CR0.ITSEN changes the value from 1 to 0, the ITS Domain begins a transition from enabled to disabled and all of the following are true:

- The transition completes in finite time.
- The transition is complete when ITS_CR0.IDLE is 1.
- All ITS events, Accepted before the transition is complete, are processed.
- All outgoing interrupt events generated for the associated IRS before the transition is complete, are Accepted by the IRS.

---

**Note**

While the ITS is transitioning from enabled to disabled, transactions arriving at the ITS will observe the ITS as being enabled or disabled. The architecture only guarantees that transactions arriving after the transition from enabled to disabled is complete to observe that the ITS is disabled.

---

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

SDNGJG When ITS_CR0.ITSEN is 1, the ITS accesses the translation structures described in 5.3 *Translation structures*. Software is expected to initialize the base address and configuration registers for the structures before enabling the ITS.

For example, software can provide a device table where all entries are invalid, or it can be a device table with valid entries containing the addresses of one or more valid ITTs, and initialize the ITS_DT_BASER and ITS_DT_CFGR to point to the device table.

SZFWSP When the ITS Domain is disabled, the ITS Domain no longer accesses translation structures and software can reclaim the memory used to hold the translation structures.

See also:

- 5.3 *Translation structures*

### 5.2.2 Interrupt event types

DCBDTC The *type* of an interrupt event generated by the ITS for an IRS is one of the following:

- SET_EDGE: The IRS should make the interrupt Pending and set its Handling mode to Edge.
- SET_LEVEL: The IRS should make the interrupt Pending and set its Handling mode to Level.
- CLEAR: The IRS should make the interrupt Idle.

RJFGLF A write to ITS_TRANSLATER generates a SET_EDGE event to the ITS.

ILKLYH The ITS only supports MSI sources that generate interrupts with edge-triggered semantics. MSI sources that are designed to signal interrupts with level-sensitive semantics using message based writes are not supported by the ITS.

RPQCKZ SET_LEVEL events are only supported for events generated by an IWB.

RTDXLR The interrupt event type generated to the IRS is the same as the ITS event type.

ILQZSR When a write updates an ITS translation structure entry, the ITS does not generate a CLEAR event as a result of the update to the translation structure. For EventIDs that represent Level interrupts and that had a valid translation before the update, if these EventIDs do not have a valid translation after the update, the corresponding interrupts may still be in the Pending state in the IRS. See 5.3 *Translation structures* for more information.

SKSFMZ Software must disable or explicitly clear the Pending state of a Level interrupt in the IRS after unmapping the event in the ITS to ensure that the interrupt is signaled to a PE. An Edge interrupt may also remain Pending after being unmapped in the ITS, but its Pending state would be cleared when the interrupt is acknowledged.

### 5.2.3 Software-generated ITS events

IQTNPX Software may generate an ITS event for an EventID and DeviceID in an ITS Domain via the following ITS registers:

- ITS_GEN_EVENT_DIDR. A write to this register selects the DeviceID of the event.
- ITS_GEN_EVENT_EIDR. A write to this register selects the EventID of the event.
- ITS_GEN_EVENTR. The TARGET_DOMAIN field in this register selects the ITS Domain where the ITS event is translated.
- A write of 1 to the R field requests the ITS to generate an ITS event of the selected type for the specified EventID and DeviceID.
- ITS_GEN_EVENT_STATUSR. A read of this register reports when generation of an ITS event is complete.

SVKLMK An event generated by writing to ITS_GEN_EVENTR may be synchronized once the write is complete to ensure that the corresponding interrupt event has been Accepted by the IRS by writing 1 to ITS_SYNCR.SYNC and polling ITS_SYNC_STATUSR.IDLE until it is 1.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 5.2.4 ITS synchronization requests

$I_{TWLSC}$ Software requests synchronization of ITS events for the ITS by writing 1 to ITS_SYNCR.SYNC.

ITS_SYNCR.SYNCALL specifies whether the synchronization request should apply to all Accepted ITS events or is only required to apply to those specified by ITS_SYNCR.DEVICE_ID.

See ITS_SYNCR for more information about the effects of a write to ITS_SYNCR.SYNC.

$R_{NLVDV}$ Writing 1 to ITS_SYNCR.SYNC requests synchronization of events that are Accepted at the time of the write and guarantees that these events are translated when ITS_SYNC_STATUSR.IDLE is 1.

Following a write to ITS_SYNCR.SYNC that requests synchronization, all of the following are true:

For each synchronized event, if the event has a valid translation, the ITS has generated an interrupt event that is Accepted by the associated IRS.

See 7.4 *Ordering and synchronization requirements for the GIC PMU* for more information about synchronization and counting events using the GIC PMU.

$S_{KDJBY}$ Software can use synchronization of events to ensure that the Pending state of Level interrupts is cleared by the IRS.

This is, for example, useful when changing the Interrupt domain association of a wire in the IWB.

See also:

- 3.2 *Communication between GIC system components*
- 4.5 *IRS synchronization requests*
- 6.2 *IWB support for multiple Interrupt Domains*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 5.3 Translation structures

$I_{KBTLK}$ The GICv5 architecture defines two translation structures which can be used by an ITS:

- The *Device Table* (DT).
- The *Interrupt Translation Table* (ITT).

$R_{SDBRM}$ All ITS translation structures are little-endian.

$R_{SVKHY}$ The ITS accesses its translation structures using Normal memory types. See [1] for more information about memory access types.

$S_{KZJVX}$ Software allocates memory for the translation structures and programs the physical address of the allocated translation structures in ITS registers. Arm expects that such allocations will typically be done from Conventional memory [1].

$I_{XTRLS}$ ITS_CR1 specifies the Shareability and Cacheability attributes for memory accesses performed by the ITS.

ITS_CR1 is read-only when the ITS is enabled.

The most recent values written to ITS_CR1 are used when the ITS accesses memory following a write that enables the ITS.

$D_{ZQBDN}$ The GICv5 architecture defines two types of translation structure entries:

- An entry in a DT, a *device table entry* (DTE).
- An entry in an ITT, an *interrupt translation table entry* (ITTE).

$D_{YMZNC}$ The VALID field of an ITS translation structure entry indicates the validity of the entry:

- When the VALID field is 1, the entry is a *valid* entry.
- When the VALID field is 0, the entry is an *invalid* entry.

$R_{MJPHW}$ The DT translates a DeviceID to an ITT base address. The DeviceID is used as an index into the DT.

$R_{XHKFY}$ The ITT translates a per-device EventID into an LPI ID, a physical/virtual qualifier, and a VM ID for virtual interrupts. The EventID is used as an index into the ITT.

$R_{TLTPV}$ An ITS Domain has a valid translation for the DeviceID and EventID of an ITS event, if all of the following are true:

- One of the following is true:
  - The DT does not use a 2-level table structure.
  - The level 1 DTE for the DeviceID is valid.
- The level 2 DTE for the DeviceID is valid.
- One of the following is true:
  - The ITT does not use a 2-level table structure.
  - The level 1 ITTE for the EventID is valid.
- The level 2 ITTE for the EventID is valid.
- One of the following is true:
  - The event is generated by a write to a register in the PAS native to the ITS Domain.
  - The event is generated by a write to a register in the Non-secure PAS, and all of the following are true:
    - The event is translated in the Realm ITS Domain.
    - L2_ITTE.DAC is `0b01` in the L2_ITTE for the EventID.

Otherwise, if any of the above conditions are false, the ITS Domain does not have a valid translation for the ITS event.

$R_{PGCPL}$ If any of the following is true, it is CONSTRAINED UNPREDICTABLE whether any events are sent to the IRS:

- The ITS Domain has more than one valid translation to the same physical INTID.
- The ITS Domain has more than one valid translation to the same virtual INTID with the same VM ID.

See also:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 11.1 *ITS Data Structures*

### 5.3.1 The Device Table (DT)

$D_{XJQMP}$ The DT structure is either linear or 2-level.

$I_{GRMKZ}$ ITS_IDR1.DT_LEVELS reports whether 2-level DT support is implemented.

$R_{WKPKD}$ The value of ITS_IDR1.DT_LEVELS is the same for all ITS Domains for an ITS in the system.

$I_{VZYWW}$ Arm strongly recommends that an implementation that supports more than 9 bits of DeviceID implements 2-level device tables.

$I_{SWCZW}$ The GICv5 architecture defines the formats of the level 1 DTE and level 2 DTE.

$I_{ZMRVW}$ The base address of the DT is stored in ITS_DT_BASER.ADDR.

$I_{VZVJN}$ When ITS_CR0.ITSEN is 1, accesses to fields in ITS_DT_BASER and ITS_DT_CFGR are **RO**.

$I_{WBLVK}$ When the DT uses a 2-level table structure, the address stored in ITS_DT_BASER.ADDR is the base address of the level 1 table.

When the DT uses a linear table structure, the address stored in ITS_DT_BASER.ADDR is the base address of the array of level 2 DTEs.

$S_{LDVFG}$ Software is responsible for allocating the DT from memory in the PAS associated with the ITS Domain where the DT is used.

$I_{KRKBH}$ The structure of the DT is controlled using the following register and translation structure fields:

- ITS_DT_CFGR.DEVICEID_BITS

  Selects how many bits of DeviceID the DT can translate. This impacts the size of the level 2 DT when using a linear structure and the size of the level 1 DT when using 2-level structure.

- ITS_DT_CFGR.STRUCTURE

  Selects if the DT uses a linear or 2-level structure.

- ITS_DT_CFGR.L2SZ:

  When using a 2-level DT, this field configures the number of DeviceID bits resolved by each level 2 DT.

- L1_SPAN:

  When using a 2-level DT, for the level 1 entry corresponding to a range of DeviceIDs, this field configures the number of DTEs in the level 2 DT.

$I_{PZYZM}$ The supported values for ITS_DT_CFGR.L2SZ are reported in ITS_IDR1.L2SZ.

$I_{NTYVG}$ The size of a level 1 DTE and a level 2 DTE is 8 bytes.

Table 5.1 shows some example DT structures:

**Table 5.1: Example DT structures.**

| STRUCTURE | DEVICEID_BITS | L2SZ | L1 size | L2 size | Maximum number of devices |
|---|---|---|---|---|---|
| Linear | 4 | - | - | 128 B | 16 |
| Linear | 9 | - | - | 4 KB | 512 |
| Linear | 13 | - | - | 64 KB | 8,192 |
| Linear | 18 | - | - | 2 MB | 262,144 |
| 2-level | 18 | 0b00, 9 bits | 4 KB | 4 KB | 262,144 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| STRUCTURE | DEVICEID_BITS | L2SZ | L1 size | L2 size | Maximum number of devices |
|---|---|---|---|---|---|
| 2-level | 27 | 0b00, 9 bits | 2 MB | 4 KB | 134,217,728 |
| 2-level | 22 | 0b01, 11 bits | 16 KB | 16 KB | 4,194,304 |
| 2-level | 26 | 0b10, 13 bits | 64 KB | 64 KB | 67,108,864 |
| 2-level | 32 | 0b10, 13 bits | 4 MB | 64 KB | 4,290,000,000 |

In the table above, the L2SZ column shows the value of ITS_DT_CFGR.L2SZ along with its interpretation in bits of DeviceID resolved by each level.

See 11.1 *ITS Data Structures* and ITS_DT_CFGR for more information.

SYMMZX The L1 size and L2 size columns in Table 5.1 indicate the required maximum physically contiguous allocation by software for the configuration. Arm expects that for large systems with many devices a 2-level structure with a large level 1 table can be allocated by software on system boot.

SFGKBW L1_DTE.SPAN field is used to limit the amount of storage required for the level 2 DTEs. For example, if the L2SZ is configured at 9 bits, and only two devices are differentiated using bit 0 of the DeviceID for the DeviceID space defined by bits N:9, then a SPAN of 1 can be used to only require that two level 2 DTEs are allocated. See Figure 5.3 for an illustration where the L2SZ is configured 9 bits and where two level 1 DTEs use different SPAN values.

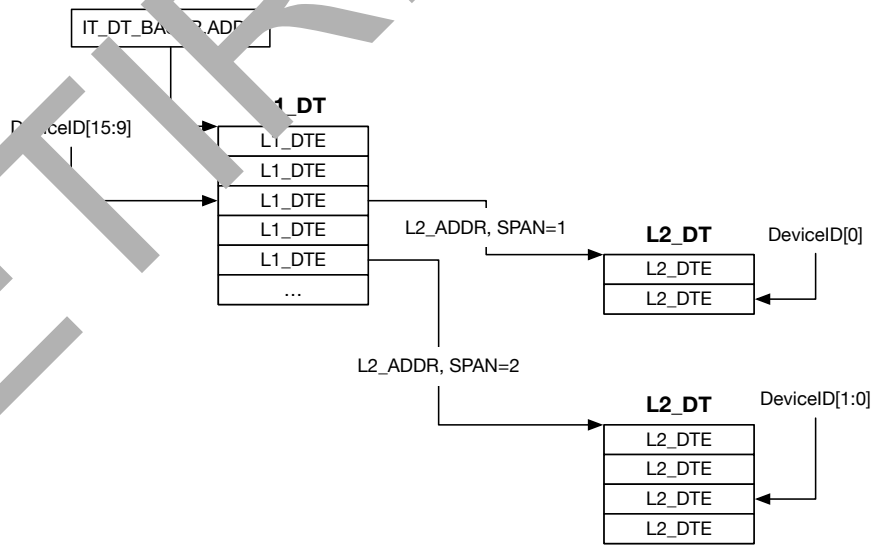



**Figure 5.3: Example device table using L1_DTE.SPAN**

SCQRVC ITS_DT_CFGR.DEVICEID_BITS determines the number of level 2 DTEs the DT contains. When using a 2-level structure, the number of DeviceID bits resolved by each level 2 table is decided by ITS_DT_CFGR.L2SZ, and software must allocate enough level 1 DTEs to cover the full DeviceID space for the DT.

DQFTBH A DeviceID is considered *invalid* if any of the following is true:

- The DeviceID is larger than (2 ^ ITS_DT_CFGR.DEVICEID_BITS) - 1.
- DeviceIDs which do not have a corresponding level 2 DTE because of how the L1_DTE.SPAN field is configured.

RHGKJK Events signaled with an invalid DeviceID are ignored and are not translated by the ITS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{XSFXB}$ A level 1 DT is aligned to the size of the level 1 table.

A level 2 DTE array is aligned to the size of the array.

See ITS_DT_BASER for more information.

$I_{CFRYB}$ When limiting a level 2 DT size by using the level 1 SPAN field, the level 2 DT address is aligned to the size of the level 2 DTE array.

$R_{XNJYG}$ When an ITS Domain is enabled, a write to any field in a valid level 2 DTE other than when setting the L2_DTE.VALID field to 0, results in CONSTRAINED UNPREDICTABLE behavior with a choice of:

- The old values are used for the device.
- The new values are used for the device.
- A combination of the old and new values is used for the device.

$I_{DHHJN}$ When the ITS Domain is enabled, changing the address of an ITT for a valid level 2 DTE, requires performing the following series of actions to avoid UNPREDICTABLE behavior:

1. Performing a write that sets the L2_DTE.VALID field to 0.
2. Ensuring that the write is visible to the ITS Domain by performing the necessary cache invalidation.
3. Writing the address and configuration fields of the new ITT to the level 2 DTE.
4. Performing a write that sets the L2_DTE.VALID field to 1.

See *5.4.2 ITS cache management for DeviceIDs* for more information about ITS device cache management.

### 5.3.2 The Interrupt Translation Table (ITT)

$D_{PPZDD}$ The ITT structure is either linear or 2-level.

$I_{RPQVS}$ ITS_IDR1.ITT_LEVELS reports whether level 1 ITT support is implemented.

$R_{TJCMV}$ The value of ITS_IDR1.ITT_LEVELS is the same for all ITS Domains in an ITS in the system.

$I_{PLTFB}$ The GICv5 architecture defines the formats of the level 1 ITTE and level 2 ITTE.

$I_{QJTXY}$ The base address of the ITT is stored in L2_DTE.ITT_ADDR.

$I_{FMQQL}$ When the ITT uses a 2-level table structure, the address stored in L2_DTE.ITT_ADDR is the base address of the level 1 table.

When the ITT uses a linear table structure, the address stored in L2_DTE.ITT_ADDR is the base address of the array of level 2 ITTEs.

$D_{CFDKB}$ A level 2 ITT *overlaps* with another level 2 ITT, if the same address for a level 2 ITTE can be reached via separate DTs of ITT.

For example, a level 2 ITT overlaps with another level 2 ITT, if any of the following are true:

- Two separate level 2 DTEs point to the same level 1 ITT.
- Two separate level 2 DTEs point to the same level 2 ITT.
- Two separate level 1 ITTEs point to the same level 2 ITT.
- The base address of a level 2 ITT is between the base and end addresses of another level 2 ITT.

$R_{XGPSP}$ If any part of either a level 1 or level 2 ITT overlaps with another ITT, the ITS behavior is CONSTRAINED UNPREDICTABLE with any of the following:

- Any ITS cache invalidation by DeviceID or EventID may not work as expected.
- ITS events may not be translated.
- The ITS may translate an ITS event resulting in sending an interrupt event to the IRS with UNKNOWN data.

Software can recover from this situation by disabling the ITS and configuring the translation structures without any overlap before re-enabling the ITS.

See *5.5 ITS memory access rules* for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$S_{YDTKM}$ Software is responsible for allocating the ITT from memory in the PAS associated with the ITS Domain where the ITT is used.

$I_{SRXXT}$ The structure of the ITT is controlled using the following fields in the ITS translation structures:

- L2_DTE.EVENTID_BITS:

  Selects how many bits of EventID the ITT can translate. This impacts the size of the level 2 ITT when using a linear structure and the size of the level 1 ITT when using 2-level structure.

- L2_DTE.ITT_STRUCTURE:

  Selects if the ITT uses a linear or 2-level structure.

- L2_DTE.ITT_L2SZ:

  When using a 2-level ITT, this field configures the number of EventID[ ] resolved by each level 2 ITT.

- L1_ITTE.SPAN:

  When using a 2-level ITT, for the level 1 entry corresponding to a [ ]ge of EventI[ ]s, this field configures the number of entries in the level 2 ITT.

$I_{HPWVL}$ The supported values for L2_DTE.ITT_L2SZ are reported [ ] ITS_ID[ ]1.L2SZ

$I_{NJHFT}$ The size of a level 1 ITTE and a level 2 ITTE is 8 bytes.

Table 5.2 shows some example ITT structures:

**Table 5.2: Example ITT structures.**

| STRUCTURE | EVENTID_BITS | L2SZ | L1 size | L2 size | Maximum number of events |
|---|---|---|---|---|---|
| Linear | 4 | - | - | 128 B | 16 |
| Linear | 9 | - | - | 4 KB | 512 |
| Linear | [ ] | - | - | 16 KB | 2,048 |
| Linear | 13 | | - | 64 KB | 8,192 |
| Linear | [ ] | - | - | 512 KB | 65,536 |
| 2-level | 11 | [ ], 9 bits | 32 B | 4 KB | 2,048 |
| 2-level | [ ] | 0b00, 9 bits | 1 KB | 4 KB | 65,536 |

In the table ab[ ]e, the L2SZ column shows the value of L2_DTE.ITT_L2SZ field along with its interpretation in bits of EventID resolved by each level.

See 11.1 *ITS Data Structures* for more information.

$S_{FYKKL}$ The L1 size and L2 size columns in Table 5.2 indicate the required maximum physically contiguous allocation by software for the configuration. Arm expects that 4KB allocations will satisfy the capabilities of most devices. For example, PCIe MSI-X supports a maximum of 2,048 events for a device. 2,048 events can either be supported with a linear 16KB table or using a 2-level structure with 4 level 1 entries each pointing to 4KB level 2 tables.

$S_{YSMWP}$ The L1_ITTE.SPAN field can be used to limit the amount of storage required for the level 2 ITT. For example, if a 2-level ITT contains entries for 544 events and the L2SZ is configured at 9 bits, then the last level 1 ITTE can configure a SPAN of 5 such that the second level 2 ITT only contains 32 entries.

$S_{ZFWKT}$ L2_DTE.EVENTID_BITS determines the number of level 2 ITTEs that the ITT contains. When using a 2-level structure, the number of EventID bits resolved by each level 2 table is decided by L2_DTE.ITT_L2SZ, and software must allocate enough level 1 ITTEs to cover the full EventID space for the ITT.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{XVTYL}$ An EventID is considered *invalid* if any of the following is true:

- The EventID is larger than (2 ^ L2_DTE.EVENTID_BITS) - 1.
- EventIDs which are not described by level 2 ITTEs because of how the L1_ITTE.SPAN field is configured.

$R_{XJGWV}$ Events signaled with an invalid EventID are ignored and are not translated by the ITS.

$I_{XRKXZ}$ If software error reporting is supported, the ITS reports if an ITS event cannot be translated due to an invalid EventID using the appropriate error code in ITS_SWERR_STATUSR.EC.

$R_{FVJTG}$ The ITTs are always aligned to the size of the table.

$I_{WBPQX}$ When limiting a level 2 ITT size by using the level 1 SPAN field, the level 2 ITT address is aligned to the size of the level 2 ITTE array.

$I_{CKRKW}$ Software may configure the ITS to translate an ITS event, generated by a write to a register in the Non-secure PAS, using the Realm ITS Domain. Such an event does not violate the separati[illegible] bet[illegible] Interrupt Domains, because the event is only translated successfully in the Realm ITS Domain if L[illegible]ITTE.DAC[illegible] `0b01` for the event.

See also:

- 5.1.1 *Supporting Realm interrupts from Non-secure wr[illegible]s*.

$I_{TSBSN}$ Software may read translation information for an EventI[illegible]nd Devi[illegible]D in an I[illegible] Domain via the following ITS registers:

- ITS_DIDR. A write to this register selects t[illegible] De[illegible]eID o[illegible]e ev[illegible].
- ITS_EIDR. A write to this register sele[illegible]s the Eve[illegible]ID of the e[illegible]ent.
- ITS_READ_EVENTR. A write of 1 to th[illegible]R[illegible]eld requests that the ITS returns translation information for the selected event.
- ITS_STATUSR. A read of [illegible]s register r[illegible]orts wh[illegible]n the translation information for the event is available.
- ITS_READ_EVENT_[illegible]ATA[illegible] A read of thi[illegible]egister provides the translation information for the event once it is available.

See also:

- ITS_DIDR.
- ITS_EIDR.
- ITS_[illegible]EVE[illegible]R.
- IT[illegible]_GEN[illegible]EVENT[illegible]T[illegible]USR.
- [illegible]TS_GEN[illegible]EVENT_E[illegible]DR.
- [illegible]TS_G[illegible]_E[illegible]_DIDR.
- IT[illegible]EAD_EVENTR.
- ITS_[illegible]AD_EVENT_DATAR.
- ITS_ST[illegible]USR.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 5.4 ITS cache management

$R_{FXXDG}$ An ITS is permitted to cache any part of the ITS translation structures at any time when they are accessible by the ITS.

$I_{KKDTR}$ The ITS caches are not considered data or system caches belonging to the memory system, meaning they are not managed by any cache maintenance instructions issued by a PE.

**Note**

This means that information from all of level 1 DTEs, level 2 DTEs, level 1 ITTEs, and level 2 ITTEs may be cached by the ITS.

$I_{NRPBV}$ An ITS is permitted to cache both valid and invalid entries of ITS translation structures.

$R_{XVTHB}$ Entries held in ITS caches are associated with the ITS Domain that contains the ITS translation structures used to derive the entry.

**Note**

This ensures isolation between ITS Domains and ensures that modifying translation structures and performing cache maintenance for one ITS Domain does not affect the behavior of another ITS Domain.

$R_{MKQFF}$ When an ITS Domain is disabled, the ITS caches contain no data from ITS translation structures associated with that ITS Domain.

$R_{JTCBT}$ Cached information from an ITS translation structure entry is not guaranteed to remain in the ITS caches.

$I_{QSGHD}$ Cached information from any ITS translation structure entry is not guaranteed to remain in the ITS caches. This means that when the ITS caches an ITS translation structure entry with different values than what is stored in memory, the ITS may naturally evict the cached entry and read the updated translation structure entry from memory, without any ITS cache invalidation operation having been performed.

$R_{HQFDL}$ An update to an ITS translation structure entry is not guaranteed to be visible to the ITS until an appropriate cache invalidation operation is completed.

$I_{GMVTV}$ When software updates the fields in an ITS translation structure entry, the ITS is permitted to cache both the old and the new entry. For as long as both entries exist in the ITS cache, each translation of a DeviceID and EventID affected by the update entry is permitted to use either the old or the new entry.

$S_{PJDQP}$ To avoid the ITS using the old translation entries, software should perform the following steps:

1. Ensure that no events are generated for that DeviceID.
2. Synchronize events Accepted by the ITS that use the old translation structures. See also 5.2.4 *ITS synchronization requests* for more information.
3. Update the fields in the ITS translation structure entry and perform the appropriate cache invalidation operation. See also 5.4.1 *ITS cache management for EventIDs* and 5.4.2 *ITS cache management for DeviceIDs* for more information.
4. Enable events for that DeviceID.

$I_{WBWHP}$ ITS cache invalidation is done for a device or an event. A device is identified by its DeviceID. An event is identified by its DeviceID and EventID.

$I_{BPDHC}$ The ITS provides cache invalidation operations that apply to any cached information from ITS translation structure entries for the specified device or event. An invalidation operation applies to the cached information from ITS translation structures irrespective of the values stored in the ITS translation structures in memory.

For example, for an event, the state of the corresponding level 2 ITTE could be as follows:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- There is cached information from a valid level 2 ITTE for that event.
- There is no cached information from the level 2 DTE for the DeviceID of the event.
- The level 2 DTE for the DeviceID of the event is invalid.

The ITS cache invalidation operation for that event invalidates the cached information from the valid level 2 ITTE even though the level 2 DTE is invalid.

$I_{LNLHV}$ ITS cache invalidation operations invalidate cached information regardless of the value of the VALID field in the cached translation structure entry.

$I_{WZQQN}$ The effects of ITS cache invalidation operations are complete when ITS_STATUSR.IDLE is 1.

$R_{ZXXJF}$ When ITS_STATUSR.IDLE transitions from 0 to 1, any ITS event that used information from ITS translation structure entries prior to the invalidate have been Accepted by the IRS.

### 5.4.1 ITS cache management for EventIDs

$I_{RPBTZ}$ An ITS cache invalidation operation for a single EventID is performed as follows:

1. The EventID is written to ITS_EIDR.EVENT_ID and the DeviceID is written to ITS_DIDR.DEVICE_ID.
2. A write to ITS_INV_EVENTR is performed with the fields set to the following:
   - ITS_INV_EVENTR.L1 is set to 0.
   - ITS_INV_EVENTR.I is set to 1.

$R_{KYPHL}$ An ITS cache invalidation operation for an EventID invalidates all cached information from the L2 ITTE corresponding to the EventID.

$I_{GLSYM}$ The ITS supports invalidating a range of EventIDs corresponding to the number of EventIDs described by a single level 2 ITT. The size of the range is selected using ITS_INV_EVENTR.ITT_L2SZ.

An ITS cache invalidation operation for a range of EventIDs is performed as follows:

1. Any EventID within the range is written to ITS_EIDR.EVENT_ID and the DeviceID is written to ITS_DIDR.DEVICE_ID.
2. A write to ITS_INV_EVENTR is performed with the fields set to the following:
   - ITS_INV_EVENTR.L1 is set to 0.
   - ITS_INV_EVENTR.ITT_L2SZ is set to the size of a level 2 ITT corresponding to the range of EventIDs that are invalidated.
   - ITS_INV_EVENTR.I is set to 1.

$R_{PBJYK}$ An ITS cache invalidation operation for a range of EventIDs invalidates all cached information from the L1 ITTE and L2 ITTEs for the DeviceID corresponding to the invalidate operation's DeviceID and for the EventID bits corresponding to the invalidate operation's EventID's bits[15:N], where N equals (9 + 2 * ITS_INV_EVENTR.ITT_L2SZ).

---

**Note**

An ITS cache invalidation for a range of EventIDs applies to cached information from that range of EventIDs, regardless of whether a linear or 2-level structure is used for the ITT when the information is cached.

---

$I_{JNKRH}$ In an invalidation operation for a range of EventIDs, for each EventID in the range, the ITS invalidates the same cached information as when an invalidate operation is performed for a single EventID. See rule KYPHL for more information.

$I_{FLZSK}$ The effects of a write to ITS_INV_EVENTR are complete when ITS_STATUSR.IDLE is 1.

See also:

- ITS_INV_EVENTR.
- ITS_DIDR.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- ITS_EIDR.
- ITS_STATUSR.

### 5.4.2 ITS cache management for DeviceIDs

$I_{TKFBL}$ An ITS cache invalidation operation for a single DeviceID is performed as follows:

1. The DeviceID is written to ITS_DIDR.DEVICE_ID.
2. A write to ITS_INV_DEVICER is performed with the fields set to the following:
   - ITS_INV_DEVICER.L1 is set to 0.
   - ITS_INV_DEVICER.EVENTID_BITS is set to the number of EventID bits to which the invalidation operation applies.
   - ITS_INV_DEVICER.I is set to 1.

$R_{NWHYF}$ An ITS cache invalidation operation for a DeviceID invalidates all of the following:

- All cached information from the L2 DTE that corresponds to the DeviceID.
- All cached information from all L1 ITTEs that correspond to the DeviceID.
- All cached information from L2 ITTEs for the DeviceID where the EventID is less than 2 ^ ITS_INV_DEVICER.EVENTID_BITS.

$I_{DWLGT}$ The ITS supports invalidating a range of DeviceIDs corresponding to the number of DeviceIDs described by a single level 2 DT.

An ITS cache invalidation operation for a range of DeviceIDs is performed as follows:

1. A DeviceID within the range is written to ITS_DIDR.DEVICE_ID.
2. A write to ITS_INV_DEVICER is performed with the fields set to the following:
   - ITS_INV_DEVICER.L1 is set to 1.
   - ITS_INV_DEVICER.I is set to 1.

$R_{FWYGP}$ An ITS cache invalidation operation for a range of DeviceIDs invalidates all cached information from the L1 DTE, L2 DTEs, L1 ITTEs, and L2 ITTEs for DeviceID bits[31:N] corresponding to the invalidate operation's DeviceID bits[31:N], where N equals (9 + 2 * ITS_DT_CFGR.L2SZ) and the effective value of EVENTID_BITS is the value reported in ITS_IDR2.EVENTID_BITS.

$I_{NTHYD}$ When a cache invalidation operation for a range of DeviceIDs is performed, the Effective value of EVENTID_BITS used to invalidate each DeviceID is the maximum implemented number of EventID bits reported in ITS_IDR2.EVENTID_BITS.

$I_{XBVDN}$ In an invalidation operation for a range of DeviceIDs, for each DeviceID in the range, the ITS invalidates the same cached information as when an invalidate operation is performed for a single DeviceID.

This means that cached information from the following translation structures is invalidated for each DeviceID when invalidating a range of DeviceIDs:

- The L2 DTE that corresponds to the DeviceID.
- All L1 ITTEs that correspond to the DeviceID.
- All L2 ITTEs that correspond to the DeviceID.

See rule NWHYF for more information.

$I_{WWQDF}$ The effects of a write to ITS_INV_DEVICER are complete when ITS_STATUSR.IDLE is 1.

$I_{QNNQG}$ When a write to a level 1 DTE sets VALID to 0, and there is cached information from a valid level 2 DTE for a DeviceID in the range covered by that level 1 DTE, the ITS may also contain cached information of the previously valid level 1 DTE.

This means that the ITS is permitted to treat the DeviceID either as valid or invalid, and to ensure the ITS treats the DeviceID as invalid, the level 1 DTE cache entries for the DeviceID range must be invalidated.

See also:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- ITS_INV_DEVICER.
- ITS_STATUSR.
- ITS_DT_CFGR.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 5.5 ITS memory access rules

$R_{BXBXJ}$ If an ITS access to any ITS translation structure occurs in PCIe address space, then the ITS is permitted to return an UNKNOWN value or terminate the access and report the error.

In this case, the ITS is permitted to drop the translation and not send any outgoing interrupt events to the IRS.

$R_{GRXYB}$ When an ITS accesses a memory location in a PAS, it relies on information stored in registers and translation structures that are accessible only within the same PAS to validate that the access is permitted.

This includes scenarios in which the ITS behavior is CONSTRAINED UNPREDICTABLE, UNPREDICTABLE, or IMPLEMENTATION DEFINED.

For example, an ITS access to a translation structure is permitted only if the memory location lies within a region defined by a base address and size held in a register or another translation structure entry, where that defining information resides in the same PAS as the access itself.

This prevents software running in a Security state from directing the ITS to access any memory location using another PAS than the one associated with the software's Security state.

$R_{BLYPY}$ An ITS does not access any memory location which is not derived from address and configuration data in the ITS registers and translation structures.

When a memory address is stored in an address field in a register or translation structure entry, and that register or entry contains a field indicating that the address is not valid, the ITS does not derive any memory location from that address field.

This includes scenarios in which the ITS behavior is CONSTRAINED UNPREDICTABLE, UNPREDICTABLE, or IMPLEMENTATION DEFINED.

$R_{LTFQT}$ When the ITS accesses an ITS translation structure, the access must be a 64-bit aligned, single-copy atomic access of at least 64 bits in size.

$I_{JXSPX}$ If the ITS experiences an external abort during a memory access to an ITS translation structure, the ITS stops the operation and reports the error.

If software error reporting is supported, the error is reported with ITS_SWERR_STATUSR.EC in the range from 0x00 to 0x04.

$R_{LSCDT}$ If the ITS experiences an external abort as part of translating an event, the translation is dropped and no outgoing interrupt event is sent to the IRS.

$R_{PGRJQ}$ If an ITS translation structure overlaps with any other ITS or IRS translation structure, the ITS behavior is UNPREDICTABLE.

The UNPREDICTABLE behavior may result in loss of interrupt configuration and state, but must not result in access to memory outside the PAS associated with the ITS Domain or in access to any memory location which is not derived from address and configuration of the DT or ITT.

For more information about the IRS translation structures, see 4.7 *The interrupt state table (IST)* and 4.9 *Virtualization data structures*.

Software can recover from this situation by performing the following sequence:

1. Disabling the ITS.
2. Making the corresponding translation structures invalid in the IRS.
3. Reconfigure the base addresses of the translation structure to avoid overlap.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 5.6 ITS support for MPAM

$I_{XZGHC}$ The Memory System Resource Partitioning and Monitoring Memory, MPAM, architecture defines per-transaction attributes that affect system behavior or the behavior of components that the transactions pass through or Completers that satisfy a transaction [1].

The ITS supports MPAM with the following additional attributes:

- Partition ID, or PARTID.
- Performance Monitoring Group, or PMG.

The PARTID and PMG are both interpreted within a PARTID space. The PARTID space used depends on the Security state associated with an ITS Domain, and the PARTID and PMG values used may be programmed independently for each ITS Domain. See [1] for more information about MPAM information bundles.

$I_{FBPMW}$ In the ITS, support for MPAM is indicated by ITS_IDR0.MPAM.

If MPAM is supported, the supported PARTID and PMG width is indicated in ITS_MPAM_IDR.{PARTID_MAX,PMG_MAX}.

$R_{FCMHZ}$ If MPAM is supported, ITS accesses to memory are associated with the PARTID and PMG programmed in ITS_MPAM_PARTID_R.{PARTID,PMG}.

$I_{THJMY}$ In systems without support for RME, the PARTID space used by the ITS is determined by the memory system attribute MPAM_NS.

In systems with support for RME, the PARTID space used by the ITS is determined by the memory system attribute MPAM_SP.

$I_{VCSKZ}$ The ITS architecture has optional support for MPAM PARTID space selection indicated by ITS_MPAM_IDR.HAS_MPAM_SP.

If ITS_MPAM_IDR.HAS_MPAM_SP is 0, each ITS Domain uses a default MPAM PARTID space.

The following table shows the MPAM PARTID space used for accesses made by the ITS Domains if ITS_MPAM_IDR.HAS_MPAM_SP is 0 and the system does not support RME:

| ITS Domain | MPAM PARTID space |
|---|---|
| Secure | Secure PARTID space |
| Non-Secure | Non-secure PARTID space |
| EL3 | Secure PARTID space |

The following table shows the MPAM PARTID space used for accesses made by the ITS Domains if ITS_MPAM_IDR.HAS_MPAM_SP is 0 and the system supports RME:

| ITS Domain | MPAM PARTID space |
|---|---|
| Secure | Secure PARTID space |
| Non-Secure | Non-secure PARTID space |
| EL3 | Root PARTID space. |
| Realm | Realm PARTID space. |

If ITS_MPAM_IDR.HAS_MPAM_SP is 1, the ITS uses the MPAM PARTID specified by ITS_MPAM_PARTID_R.MPAM_SP.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{BJQFG}$ If an ITS without support for MPAM is integrated in a system that supports MPAM, the PARTID and PMG used for each ITS Domain is IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 5.7 ITS support for Memory Encryption Contexts

$I_{PRBCL}$ The Memory Encryption Contexts feature, FEAT_MEC, provides finer-grained memory encryption contexts, within the Realm physical address space, to be assigned to Realms, with policy controlled by Realm EL2 [1].

$G_{ZHPQC}$ The GICv5 architecture ensures that an ITS can function correctly in systems with support for Memory Encryption Contexts (MEC).

$I_{YQGHF}$ In a system with support for MEC, the PEs write ITS translation structures to memory and these are read by the ITS.

In such a system, all ITS Realm translation structures are managed and configured by Realm EL2, and virtualized software running at EL1 is not expected to have direct access to the ITS configuration registers or translation structures. Therefore, the ITS only supports configuration of a single MECID used for all ITS memory accesses to the Realm PAS. This allows software running at Realm EL2 to write to the ITS translation structures and allows the ITS to read the translations.

$I_{VRVQV}$ In the ITS, support for Memory Encryption Contexts (MEC) is indicated in ITS_IDR.MEC for the Realm ITS Domain. If the MEC feature is supported, the supported MECID width is indicated in ITS_MEC_IDR.MECIDSIZE for the Realm ITS Domain.

Arm strongly recommends that the MECID bit width supported by the ITS matches or exceeds the width supported by the PEs in the system.

$R_{XKTWP}$ If the MEC feature is supported, ITS accesses to memory are associated with a MECID that identifies the Memory Encryption Context of the access.

$R_{XTVVQ}$ Accesses made by the ITS for Secure, Non-secure and Root PA spaces are issued with the default MECID of zero.

$I_{FPSSV}$ Accesses made by the ITS to Realm PA spaces are associated with the global Realm PAS ITS MECID programmed in ITS_MEC_MECID_R.MECID.

$R_{DGJXQ}$ If an ITS without support for the Realm ITS Domain is integrated in a system that supports MEC, all ITS accesses for that ITS are treated as having the default MECID of zero.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 5.8 ITS support for software error reporting

$I_{FWKKS}$ The ITS specifies a mechanism to report errors because of incorrect programming.

$R_{FBVQN}$ The ITS detects software errors while processing ITS events. Otherwise, the ITS does not detect software errors.

$R_{RLHPC}$ Software error codes that refer to incoming events apply to incoming events generated by all supported mechanisms. See Chapter 5 *Interrupt translation service (ITS)* for more information about the supported mechanisms for incoming events.

$R_{RLJNX}$ ITS support for software error reporting is optional.

$I_{HKRKL}$ ITS_IDR0.SWE reports whether software error reporting is supported.

$R_{QRYDD}$ The value of ITS_IDR0.SWE is the same for all ITS Domains in an ITS in the system.

$I_{FCPZL}$ When ITS_IDR0.SWE is 1, the ITS uses the ITS_SWERR_STATUSR.IMP_EC to report any IMPLEMENTATION DEFINED errors detected by the ITS.

$I_{ZGMTB}$ ITS_SWERR_STATUSR.EC is 0 when an IMPLEMENTATION DEFINED error is reported by the ITS.

$I_{ZNSVM}$ When L2_DTE.DSWE is 0, the ITS does not report errors for the following error codes in ITS_SWERR_STATUSR.EC:

- 0x03: Failed lookup of L1_ITTE due to an external abort.
- 0x04: Failed lookup of L2_ITTE due to an external abort.
- 0x06: An incoming event could not be translated because L2_DTE.VALID is 0.
- 0x07: An incoming event could not be translated because L1_ITTE.VALID is 0.
- 0x08: An incoming event could not be translated because L2_ITTE.VALID is 0.
- 0x0A: An incoming event could not be translated because the EventID > (2 ^ L2_DTE.EVENTID_BITS) - 1.
- 0x0C: An incoming event could not be translated because the EventID exceeds the L1_ITTE.SPAN.
- 0x0D: An incoming event could not be translated because the event is associated with the Non-secure Interrupt Domain and L2_DTE.DAC = 0.

$S_{FPVFM}$ When a device is assigned to a VM, the Hypervisor can set L2_DTE.DSWE to 0 in the L2 DTE of the device to mask reporting of software errors caused by misconfiguration of the device by VM.

$I_{ZKJKN}$ Software error information can be read in an ITS Domain via the following ITS registers:

- ITS_SWERR_STATUSR.
- ITS_SWERR_SYNDROMER0.
- ITS_SWERR_SYNDROMER1.

$I_{YSSCL}$ When a software error is reported in the ITS Domain, the value of ITS_SWERR_STATUSR.V is 1. Otherwise, no software error is reported in the ITS Domain and fields in this register are UNKNOWN.

$R_{JGVHM}$ When software error reporting is implemented, the ITS reports a detected error using any of the defined error codes or sets ITS_SWERR_STATUSR.OF.

The ITS is not permitted to silently ignore an error that could be reported.

$I_{PCPFQ}$ When more than one error code can be used to describe a detected error, it is IMPLEMENTATION DEFINED which error code is used.

The architecture does not define a priority between error codes that can occur at the same time.

$I_{PVQYF}$ When the value of ITS_SWERR_STATUSR.V is 1, all of the following are true in that ITS Domain:

- ITS_SWERR_STATUSR.EC specifies the fault that caused the software error.
- ITS_SWERR_STATUSR.S0V indicates whether ITS_SWERR_SYNDROMER0 contains valid error syndrome information.
- ITS_SWERR_STATUSR.S1V indicates whether ITS_SWERR_SYNDROMER1 contains valid error syndrome information.
- ITS_SWERR_STATUSR.OF indicates whether multiple software errors were detected.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{ZBHQQ}$ When ITS_SWERR_STATUSR.S0V is 1, ITS_SWERR_SYNDROMER0 reports the following information for the software error:

- The DeviceID and EventID of the incoming event that resulted in the software error.
- The Interrupt Domain the incoming event is associated with.

$I_{BBZNG}$ When ITS_SWERR_STATUSR.S1V is 1, ITS_SWERR_SYNDROMER1 reports the address of the ITS translation structure associated with the software error.

$R_{LKLQG}$ For each reported error, the values of ITS_SWERR_STATUSR.{S0V,S1V} are IMPLEMENTATION DEFINED and set independently.

$S_{NZTWH}$ Arm recommends that software performs the following sequence of operations to either clear the last error, or detect if new errors were reported:

1. Read ITS_SWERR_STATUSR and determine which fields need to be cleared to zero.
2. In a single-copy atomic write to ITS_SWERR_STATUSR:
    1. Write ones to all the W1C fields that are nonzero in the read value.
    2. Write zero to all the W1C fields that are zero in the read value.
    3. Write zero to all the RW fields.
3. Read back ITS_SWERR_STATUSR after the write. If the value read back is the same as the value that was written, all W1C fields that were non-zero are cleared, and no new errors were reported. Otherwise, one or more new errors were reported.

See also:

- ITS_SWERR_STATUSR.
- ITS_SWERR_SYNDROMER0.
- ITS_SWERR_SYNDROMER1.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 6
# Interrupt Wire Bridge (IWB)

RETIRED

$I_{FJFTJ}$ This section describes the architecture of an IWB. The IWB detects changes to the state of input wires and signals an ITS. The ITS generates ITS events from the signals and translates them into interrupt events that are forwarded to an IRS.

$R_{LMVTM}$ A system is permitted to have zero, one, or many IWB instances.

$R_{FYRGC}$ Each IWB is associated with a single ITS in the system.

$D_{YQYMC}$ Every input wire connected to the IWB is *asserted* or *de-asserted*.

$R_{GXTZZ}$ The mechanism for signaling an input wire asserted or de-asserted is IMPLEMENTATION DEFINED.

$I_{HBSZB}$ For example, a wire connecting a device that signals *interrupt events* to the IWB may assert the input wire whenever the device signals an event and de-assert the input wire at any time after signaling the event. Only the assertion (and not the de-assertion) of the input wire matters for such a device.

As another example, a wire connecting a device signals *interrupt state* to the IWB may keep the input wire asserted to indicate a state in the device and keep the wire de-asserted to indicate a different state in the device. The input wire would be asserted and de-asserted when the device transitions between such states.

$I_{DXLSJ}$ The number of implemented wire control registers IWB_WTMR<n>, IWB_WDOMAINR<n>, and IWB_WENABLER<n> is reported in IWB_IDR0.IW_RANGE.

$I_{LTGCB}$ The maximum number of implemented wires for a single IWB instance is 65536.

$R_{MMWGD}$ The number of implemented wires by an IWB is less than or equal to (IWB_IDR0.IW_RANGE + 1) * 32.

$D_{RLQKQ}$ Each input wire to an IWB can be uniquely identified by its *input wire index* which is a number from 0 through (IWB_IDR0.IW_RANGE + 1) * 32 - 1.

$I_{ZQSGD}$ Accesses to wire control fields for wires that are not implemented in the wire control registers are RAZ/WI.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{VKMRC}$ The IWB is identified with a DeviceID that is unique in the DeviceID namespace of the associated ITS.

All wire events signaled to the ITS from the same IWB are signaled using the same DeviceID.

$R_{RMRTF}$ An input wire uses the input wire index as the EventID when signaling an event to the ITS.

$I_{MBFPR}$ Each input wire is signaled with an EventID that is unique across all input wires implemented by the IWB.

$I_{QHCJD}$ Arm strongly recommends implementing wires contiguously to avoid the need to allocate a sparse ITT for the ITS.

$S_{HYHGF}$ The relationship between wires and the connected interrupt source is available to system software in firmware tables.

$R_{RVNRP}$ The mechanism used by the IWB to communicate events to the ITS is IMPLEMENTATION DEFINED.

$I_{TVRKT}$ IWB_CR0.IWBEN controls whether the IWB is enabled or disabled.

$I_{RQJMN}$ The Trigger mode of each wire connected to an IWB is configured to be either edge-triggered or level-sensitive.

$R_{WLHDQ}$ When the IWB is enabled, events are generated according to the input wire settings:

- If an input wire is enabled and configured as edge-triggered, the IWB generates a SET_EDGE event when the input wire becomes asserted.
- If an input wire is enabled and configured as level-sensitive, the IWB generates a SET_LEVEL event when the input wire becomes asserted and a CLEAR event when the level becomes de-asserted.
- If an input wire is disabled, the IWB does not generate any events as a result of the wire becoming asserted or de-asserted.

When the IWB is disabled, it generates no events to the ITS.

$R_{MXGNY}$ When the IWB generates an event as a result of a wire being asserted or de-asserted, the event is generated in finite time.

$R_{PKSVB}$ For an edge-triggered input wire, if the wire is asserted multiple times without the IWB having sent an event to the ITS, the IWB is permitted to only generate a single SET_EDGE event and send it to the ITS.

Once an event is generated and sent to the ITS, when the wire is subsequently asserted, the IWB generates a new SET_EDGE event in finite time.

$I_{HTKHT}$ For example, when a wire that is configured as edge-triggered goes through the following transition, the IWB is permitted to generate only a single SET_EDGE event:

1. The wire is asserted
2. The wire is de-asserted
3. The wire is asserted

$R_{PHYZS}$ If an input wire is configured as level-sensitive, the IWB is permitted to discard the SET_LEVEL and CLEAR events when the wire is asserted and de-asserted before the IWB sends any event to the ITS.

$I_{HTBRD}$ For example, for a wire that is configured as level-sensitive and goes through the following transition, the IWB is permitted not to generate any events to the ITS:

1. The wire is asserted
2. The wire is de-asserted

$R_{GJJZH}$ For an edge-triggered input wire, when the IWB or the input wire is disabled, the IWB is permitted to detect that the input wire is asserted and generate a SET_EDGE event later when the IWB and the input wire are enabled.

If the input wire's Trigger mode is updated from edge-triggered to level-sensitive when the IWB or input wire is disabled, the IWB does not generate any SET_EDGE events when the IWB and the input wire are subsequently enabled.

$I_{BGSBG}$ Changing the value of IWB_CR0.IWBEN does not affect the values in other IWB registers.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

RVWTHK When a write to IWB_CR0.IWBEN changes the value from 0 to 1, the IWB begins a transition from disabled to enabled.

The transition from disabled to enabled is complete when IWB_CR0.IDLE is 1.

When the transition from disabled to enabled is complete, the IWB processes the following events:

For wires where IWB_WTMR<n>.TM<x> is 1 (level-sensitive) and IWB_WENABLER<n>.WEN<x> is 1:

- If the wire is asserted when the transition is complete, the IWB generates a SET_LEVEL event.
- If the wire is de-asserted when the transition is complete, the IWB generates a CLEAR event.

For wires where IWB_WTMR<n>.TM<x> is 0 (edge-triggered):

- The IWB does not generate any events for edge-triggered input wires when the IWB becomes enabled.

IXFJKK If a level-sensitive input wire becomes asserted or de-asserted during the IWB transitions from disabled to enabled, the IWB either generates a single event corresponding to the final state of the input wire or generates multiple events for the input wire where the last event corresponds to the final state of the input wire.

RDRQYB When a write to IWB_CR0.IWBEN changes the value from 1 to 0, the IWB begins a transition from enabled to disabled.

When the IWB is transitioning from enabled to disabled, the IWB is not guaranteed to detect when input wires are asserted or de-asserted.

The transition from enabled to disabled is complete when IWB_CR0.IDLE is 1.

When the transition from enabled to disabled is complete, the IWB has generated a CLEAR event for every level-sensitive wire where it had sent a SET_LEVEL event and no corresponding CLEAR. These events are Accepted by the ITS when the transition is complete.

---

**Note**

If the assignment of an input wire to an Interrupt Domain is changed when IWB_CR0.IDLE is 0, and before a corresponding CLEAR event is Accepted by the ITS, the IWB does not guarantee that a CLEAR event is sent or Accepted by the ITS when IWB_CR0.IDLE is 1.

See *6.2 IWB support for multiple Interrupt Domains* for more information about assigning input wires to Interrupt Domains.

---

RXCTCW If the IWB is enabled or disabled when IWB_WENABLE_STATUSR.IDLE is 0 for any PAS, it is CONSTRAINED UNPREDICTABLE whether the IWB generates events for level-sensitive wires affected by the write to ongoing write to IWB_WENABLER<n>.

See *6.1 IWB wire control registers* for more information about individually enabling and disabling wires.

RXDGSG Following a write to IWB_CR0.IWBEN, when IWB_CR0.IDLE is 1, all events generated as a result of the write are Accepted by the ITS.

SXWPYQ When the IWB is reset, software is expected to configure wired interrupts using a sequence similar to the following:

- Software executing in a Security state that can access the MPPAS of the IWB performs the following steps:
  1. Individually disable all input wires to the IWB.
  2. Initialize the Interrupt Domain association for the input wires in the IWB.
  3. Enable the IWB.
- Software executing in any Security state performs the following steps to configure a wired interrupt in an Interrupt Domain that the input wire is assigned to:
  1. Enable and configure the IRS, including providing the necessary data structures in memory.
  2. Enable and configure the associated ITS, including providing the necessary data structures in memory.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

3. Configure a valid translation in the ITS for the DeviceID and EventID associated with the IWB and input wire.
4. Configure the input wire in the IWB as edge-triggered or level-sensitive.
5. Individually enable the wire in the IWB.

See also:

- Chapter 4 *Interrupt routing service (IRS)*
- 4.8.2 *Physical SPIs*
- Chapter 5 *Interrupt translation service (ITS)*

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 6.1 IWB wire control registers

DFKXGT The IWB allows software to control the behavior of the IWB for each input wire through the following *wire control registers*:

- IWB_WENABLER<n>: Enables and disables individual wires.
- IWB_WTMR<n>: Configures if individual wires are edge-triggered or level-sensitive.
- IWB_WDOMAINR<n>: Selects the Interrupt Domain that the individual wires are assigned to.

DWLWFZ The following terms are used to indicate programming of a single wire as opposed to a configuration setting which applies to all of the IWB:

- *Individually enabling* a wire refers to a write to IWB_WENABLER<n>.WEN<x> that updates the value from 0 to 1.
- *Individually disabling* a wire refers to a write to IWB_WENABLER<n>.WEN<x> that updates the value from 1 to 0.

RSDJBH When individually enabling one or more wires, the IWB behaves as follows:

- For wires where IWB_WTMR<n>.TM<x> is 1 (level-sensitive):
  - If the wire is asserted at the time of the write to IWB_WENABLER<n>, the IWB generates a SET_LEVEL event.
  - If the wire is de-asserted at the time of the write to IWB_WENABLER<n>, the IWB generates a CLEAR event.
- For wires where IWB_WTMR<n>.TM<x> is 0 (edge-triggered):
  - The IWB does not generate any events for the wire.

When individually disabling one or more wires, the IWB behaves as follows:

- For wires where IWB_WTMR<n>.TM<x> is 1 (level-sensitive):
  - If a SET_LEVEL event was previously generated for that wire without a corresponding CLEAR event, the IWB generates a CLEAR event.
- For wires where IWB_WTMR<n>.TM<x> is 0 (edge-triggered):
  - The IWB does not generate any events for the wire.

Writing to IWB_WENABLER<n> when IWB_CR0.IWBEN is 0 has no effect other than changing the values stored in IWB_WENABLER<n>.

IWLHND The effects of a write to IWB_WENABLER<n> are complete when a read using the same PAS as the write of IWB_WENABLE_STATUSR.IDLE returns 1.

RHJXNH When the effects of a write to IWB_WENABLER<n> are complete, all of the following are true:

- Each input wire affected by the write becomes either disabled or enabled.
- All CLEAR events generated as a result of individually disabling wires are Accepted by the ITS.

---

**Note**

The IWB is required to generate a CLEAR event as a result of individually disabling a level-sensitive wire. The effects of the write to IWB_WENABLER<n> are not complete until the CLEAR events are accepted by the ITS.

The IWB is not required to generate any events and ensure that they are Accepted by the ITS before IWB_WENABLE_STATUSR.IDLE is 1 when individually enabling a wire.

---

SDDQDS The IWB is required to generate CLEAR events when individually disabling a wire to support the following use case:

- To re-sample the level of a level-sensitive interrupt, software can rely on a CLEAR being sent when the effects of individually disabling a wire are complete, and subsequent SET_EDGE or SET_LEVEL events

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

being sent in finite time.
- Software may remove a mapping for an event in the ITS and ensure that the ITS does not report failed translations for that wire due to events being sent by the IWB after the wire has been disabled.

---

**Note**

If the assignment of an input wire to an Interrupt Domain is changed when IWB_WENABLE_STATUSR.IDLE is 0 due to a write to IWB_WENABLER<n> that disables that wire, and before a corresponding CLEAR event is Accepted by the ITS, the IWB does not guarantee that a CLEAR event is sent or Accepted by the ITS when IWB_WENABLE_STATUSR.IDLE is 1.

See *6.2 IWB support for multiple Interrupt Domains* for more information about assigning input wires to Interrupt Domains.

---

$S_{TFNWQ}$ Software can poll IWB_WENABLE_STATUSR.IDLE until it is 1, using the same PAS as was used to individually disable the wire, to ensure that all events for a wire that has been disabled are Accepted by the ITS and that no further events will be generated for that wire.

Software can rely on this functionality combined with an INV synchronization request to ensure that all events are translated by the ITS before unmapping the event to avoid spurious translation errors being logged by the ITS.

$I_{VXFZX}$ Arm strongly recommends that a translation exists in the ITS for the wire before individually enabling the wire, and that the translation is not changed or made invalid in the ITS while the wire is individually disabled.

Further, Arm strongly recommends that a wire is individually disabled before changing the Interrupt Domain that the wire is assigned to. See *6.2 IWB support for multiple Interrupt Domains* for more information.

$R_{HWHSV}$ It is IMPLEMENTATION DEFINED whether software is permitted to program the Trigger mode of individual wires.

If software is not permitted to program the Trigger mode, any access to the Trigger mode of individual wires is **RO**.

$I_{YQYHL}$ The permission to program the Trigger mode is expressed by the pseudocode function `IsWireConfigRO()`.

$R_{MQFZD}$ Writing to IWB_WMODR<n>.TM<x> when any of the following are true results in a CONSTRAINED UNPREDICTABLE behavior:

- The wire is enabled.
- IWB_WENABLE_STATUSR.IDLE is 0 due to a write to IWB_WENABLER<n> affecting that wire.

The CONSTRAINED UNPREDICTABLE behavior is one of the following:

- The write is IGNORED and the configuration is not updated.
- The configuration is updated and no events are generated as a result of changing the configuration.
- The configuration is updated and the IWB generates a SET_EDGE, SET_LEVEL, or CLEAR event to the ITS.

$S_{TQWXV}$ Software is expected to configure the wire to be edge-triggered or level-sensitive when the wire is individually disabled.

$I_{BNWSF}$ Software may *resample* a wire by writing the input wire index to IWB_WRESAMPLER.

$R_{QSYNQ}$ A wire can only be resampled by writing the input wire index to IWB_WRESAMPLER using one of the following PA spaces:

- The MPPAS of the IWB.
- The PAS associated with the Interrupt Domain that the wire is assigned to.

Writing the input wire index to IWB_WRESAMPLER using any other PAS has no effects.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$R_{HFMBM}$ Resampling an enabled wire has the following effects:

- For wires where IWB_WTMR<n>.TM<x> is 1 (level-sensitive), all of the following are true:
  - If the wire is asserted, the IWB generates a SET_LEVEL event.
  - If the wire is de-asserted, the IWB generates a CLEAR event.
- For wires where IWB_WTMR<n>.TM<x> is 0 (edge-triggered), all of the following are true:
  - If the wire is asserted, the IWB generates a SET_EDGE event.
  - If the wire is de-asserted, the IWB generates no events.

Resampling a disabled wire has no effects and generates no events.

$R_{RKLJP}$ Events generated as a result of resampling a wire are generated in finite time.

$R_{MQVNP}$ If a wire is being individually enabled or disabled at the same time that a wire is resampled, it is CONSTRAINED UNPREDICTABLE whether the resample has any effects.

See also:

- 4.5 *IRS synchronization requests*
- 5.2.4 *ITS synchronization requests*
- 10.4 *IWB register frames*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 6.2 IWB support for multiple Interrupt Domains

$R_{ZDJNJ}$ An IWB only implements support for Interrupt Domains supported by PEs in the system.

It is possible for an IWB to implement a subset of the Interrupt Domains that the PEs support.

$I_{WYBFB}$ For a system with more than one IWB, it is possible for each IWB to implement a different set of Interrupt Domains.

The Interrupt Domains implemented by an IWB are reported in IWB_IDR0.INT_DOMS.

$R_{ZLWKJ}$ The MPPAS of an IWB is determined from the subset of Interrupt Domains implemented by the IWB as follows:

| Implemented Interrupt Domains | IWB MPPAS |
|---|---|
| Non-secure | Non-secure |
| Secure | Secure |
| Non-secure, Secure, EL3 | Secure |
| Non-secure, Realm, EL3 | Root |
| Non-secure, Realm, Secure, EL3 | Root |

$I_{VTYPK}$ For example, if an IWB implements the Non-secure, Secure, and EL3 Interrupt Domains, the MPPAS of the IWB is the Secure PAS. If the PEs in the system implement FEAT_RME, the MPPAS of the PEs is the Root PAS. Software executing in either Root state or Secure state can access the MPPAS of the IWB.

$R_{ZGJQC}$ The ITS associated with the IWB provides an ITS Domain for each Interrupt Domain implemented by the IWB.

$D_{TDVQF}$ Each input wire connected to the IWB is *assigned* to an Interrupt Domain.

$R_{QWBDZ}$ The assignment of a wire to an Interrupt Domain is configured in IWB_WDOMAINR<n>.

For each wire, the assignment of a wire to an Interrupt Domain is either fixed or software programmable.

$S_{MMLFL}$ The assignment of a wire to an Interrupt Domain is either fixed or software programmable. Software checks the configuration by attempting to assign a new Interrupt Domain for the wire in IWB_WDOMAINR<n> using the IWB MPPAS and reading back the value once IWB_WDOMAIN_STATUSR.IDLE is 1. If the value returned has not been updated, the wire assignment to the Interrupt Domain is fixed. Otherwise, the wire assignment is software programmable.

$I_{NZYYK}$ The Interrupt Domains supported by the IWB are reported in IWB_IDR0.INT_DOMS.

$R_{GFGYY}$ When the IWB communicates an event related to a wire to the ITS, the Interrupt Domain of the event is the Interrupt Domain that the wire is assigned to.

$I_{VGSJN}$ To provide isolation between Interrupt Domains, the configuration of an input wire is restricted based on the PAS used to write to IWB wire registers.

$I_{PVVMN}$ IWB_WDOMAINR<n> is only accessible in the MPPAS of the IWB.

For any other PAS, the registers are RAZ/WI.

$I_{VFDHJ}$ The fields in IWB_WENABLER<n> and IWB_WTMR<n> are modified only through one of the following PA spaces:

- The MPPAS of the IWB.
- The PAS associated with the Interrupt Domain that the wire is assigned to.

$I_{HTBXZ}$ For example, in an implementation supporting all four Interrupt Domains, an input wire configured to be assigned to the Secure Interrupt Domain can only be configured and enabled when writing to the MMIO registers using either the Secure or Root PAS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

$I_{FTMVV}$ The effects of a write to IWB_WDOMAINR<n> are complete when IWB_WDOMAIN_STATUSR.IDLE is 1.

$R_{RXCQB}$ When a write to IWB_WDOMAINR<n> is complete, for each wire that was affected by the write, the wire is assigned to the new Interrupt Domain.

Events generated after the write to IWB_WDOMAINR<n>, but before the write completes, for wires affected by the write, are either signaled in the old Interrupt Domain or the new Interrupt Domain of the wires.

$I_{YZCLL}$ The IWB does not generate any events as a result of a write to IWB_WDOMAINR<n>.

$R_{BMZMZ}$ If the IWB is being enabled or disabled at the same time that a wire is being assigned to a new Interrupt Domain, the IWB is not required to generate any events for that wire as a result of enabling or disabling the IWB.

See Chapter 6 *Interrupt Wire Bridge (IWB)* for more information about enabling and disabling the IWB.

$R_{TXFJS}$ If a wire is being individually enabled or disabled at the same time that a wire is being assigned to a new Interrupt Domain, the IWB is not required to generate any events for that wire as a result of individually enabling or disabling the wire.

See 6.1 *IWB wire control registers* for more information about individually enabling and disabling wires.

$S_{XWKVY}$ Software can use the following sequence to change the assignment of a wire from an old Interrupt Domain to a new Interrupt Domain:

1. Software in the old Interrupt Domain performs the following actions:
    1. Disable the interrupt in the IRS where the wire is signaled.
    2. Individually disable the wire in the IWB.
    3. Wait until IWB_WENABLE_STATUSR.IDLE is 1 to ensure that no further events will be generated for the wire.
    4. Perform a synchronization request in the ITS and wait until it has completed.
    5. Unmap the event from the ITS and invalidate the required ITS caches.
    6. Request that software running in a Security state that can access the MPPAS of the IWB, assigns the input wire to the new Interrupt Domain.
2. Software in the Security state that can access the MPPAS of the IWB performs the following actions:
    1. Writes to IWB_WDOMAINR<n>.WDDOM<x> to assign the wire to the new Interrupt Domain.
    2. Waits until IWB_WDOMAIN_STATUSR.IDLE is 1.
3. Software in the Security state that can access the MPPAS of the IWB notifies software in the new Interrupt Domain that the wire has been re-assigned.
4. Software in the new Interrupt Domain performs the following actions:
    1. Allocate and configure an interrupt in the IRS.
    2. Map the wire event in the ITS for the new Interrupt Domain.
    3. Individually enable the wire in the IWB.

See also

- 10.2 *IRS register frames*
- 10.4 *IWB register frames*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 7
# GIC Performance Monitoring Unit (PMU)

RETIRED

$D_{WQRHY}$ A GICv5 system may implement one or more *GIC Performance Monitoring Units (PMU)* that counts events for one or more GICv5 system components.

$R_{GSJJJ}$ A GIC PMU is an implementation of the *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10].

$I_{CXHZH}$ The GIC PMU registers for each GIC PMU are accessible in a separate 64KB aligned GIC_PMU_FRAME.

$R_{QZHLT}$ The base address of the GIC_PMU_FRAME is IMPLEMENTATION DEFINED.

$S_{BPXJG}$ Arm expects firmware describes the base address of ITS PMU register frames to system software using firmware data structures.

$R_{VGHTV}$ For a GIC PMU, monitor 0 is 64 bits. The sizes of all other monitors are IMPLEMENTATION DEFINED. This means that PMCFGR.SIZE is `0b111111`.

$S_{HTCFV}$ Arm expects that monitor 0 is used to count GIC cycles.

$D_{LTMWH}$ A GIC PMU counts events from GIC components. This is referred to as the *agent being monitored* in the *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10]. The agent being monitored is one of the following:

- The GIC PMU counts events for a single ITS.
- The GIC PMU counts events for a single IRS.
- The GIC PMU counts events for a single IRS and all of its associated ITSs.

Figure 7.1 shows different implementation options for GIC PMUs in a GICv5 system.

$R_{FKJCW}$ When a GIC PMU counts events for a system component that supports multiple Interrupt Domains, the GIC PMU counts events for all the supported Interrupt Domains.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

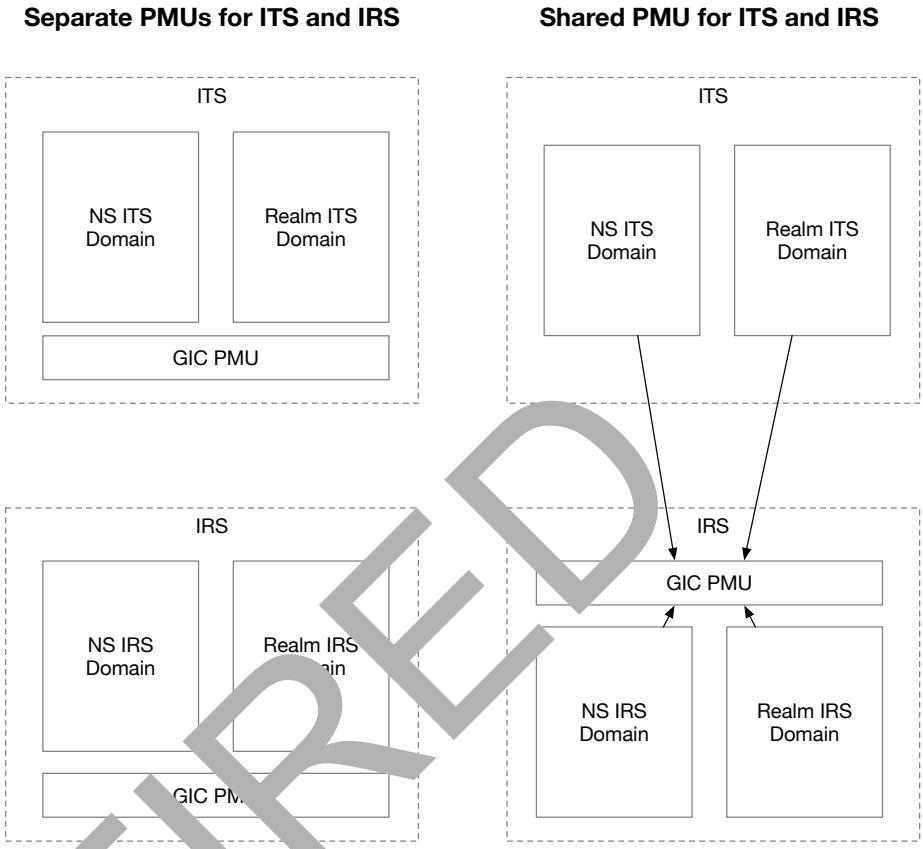



**Figure 7.1: GIC PMU implementation options**

RETIRED

**Note**

The GIC PMU implementation options are shown using only the Non-secure and Realm Interrupt Domains for illustrative purposes. The GIC PMU is also supported for the Secure and EL3 Interrupt Domains.

$I_{HLZWW}$ GIC_PMIDR reports whether the GIC PMU counts events for an IRS.

GIC_PMIDR0.ITS_PMU reports whether the GIC PMU counts events for one or more ITSs.

$R_{BMCHW}$ GIC_PMIDR {IRS_PMU, ITS_PMU} is not permitted to be {0, 0}.

See also:

- GIC_PMU_FRAME, GIC PMU register frame.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 7.1 CoreSight PMU extensions

$R_{FVCHK}$ A GIC PMU implements the 64-bit programmers' model extension defined in [10].

$I_{SCHGW}$ A GIC PMU implements the direct access to enables and overflows extension because it is always implemented when the 64-bit programmers' model extension is implemented.

$R_{TGDSP}$ A GIC PMU implements the observability and access control extension defined in [10] if, for any of the components in the agent being monitored, all of the following are true:

- Multiple Interrupt Domains are supported.
- Any of the following are true:
  - The agent supports the Secure Interrupt Domain.
  - The agent supports the EL3 Interrupt Domain.

Otherwise, it is IMPLEMENTATION DEFINED whether the GIC PMU implements the observability and access control extension.

$R_{SKSTY}$ If a GIC PMU does not implement the observability and access control extension, it is IMPLEMENTATION DEFINED which physical address space is used to access the GIC PMU registers.

$I_{JMZMR}$ GIC_PMIDR0.OACE reports whether the observability and access control extension is implemented.

$R_{QZCCM}$ When the Observability and access control extension is implemented, all of the following are true:

- It is IMPLEMENTATION DEFINED whether PMSCR and PMROOTCR are implemented in the GIC PMU register frame or at an IMPLEMENTATION DEFINED location.
- PMSCR is only accessible in the MPPA for the system.

$R_{PMHYN}$ When the Observability and access control extension is implemented, and the agent being monitored supports the Realm interrupt domain, all of the following are true:

- The configuration of PMSCR.NSRA has no impact on the access controls to the GIC PMU configuration registers.
- The configuration of PMSCR.NSPMI has no impact on physical address space used for message-signaled interrupts from the GIC PMU.
- The following fields are added to PMROOTCR, Root and Realm Control Register:

**RA, bits [1:0]** Register Access

This field determines the physical address space for which register access is enabled for the PMU.

| RA | Description |
|---|---|
| 0b000 | Root register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI. |
| 0b001 | Root and Realm register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI. |
| 0b010 | Root and Secure register access is enabled. Access from other address spaces is disabled, meaning accesses to all PMU registers are RAZ/WI. |
| 0b011 | All access is enabled. |

Other values are reserved.

For the CoreSight management registers, 0xFA8 to 0xFFC, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by this field.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

The reset value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

**RMSI, bits [11:10]** Root control for MSI PA space

When PMCFGR.MSI is 1, this field determines the physical address space used for MSIs.

| RMSI | Description |
|---|---|
| 0b000 | MSIs are generated using the Root PAS. If PMROOTCR.RA is not `0b000`, this value results in the Non-secure PAS being used for MSIs. |
| 0b001 | MSIs are generated using the Realm PAS. If PMROOTCR.RA is `0b010`, this value results in the Non-secure PAS being used for MSIs. |
| 0b010 | MSIs are generated using the Secure PAS. If PMROOTCR.RA is `0b0x1`, this value results in the Non-secure PAS being used for MSIs. |
| 0b011 | MSIs are generated using the Non-secure PAS. |

Other values are reserved.

The reset value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

$R_{TSZTF}$ It is IMPLEMENTATION DEFINED whether GIC PMU implements the Freeze on overflow extension defined in [10].

$R_{KLJSW}$ A GIC PMU does not implement the Half-word non-debug extension defined in [10]. For GIC PMU, PMCFGR.HDBG is 0.

$R_{XHQVD}$ A GIC PMU does not implement the Fixed-function cycle counter extension defined in [10].

$R_{VQYFY}$ A GIC PMU does not implement the Monitor group extension defined in [10]. For GIC PMU, PMCFGR.NCG is 0.

$R_{CZMLC}$ A GIC PMU does not implement the Counter chaining extension defined in [10].

$R_{DSXHC}$ A GIC PMU does not implement the event counter threshold and edge detection extensions defined in [10].

$I_{SKNKR}$ The *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10] defines Reusable event filter definitions.

When a GIC PMU implements the observability and access control extension, the GIC PMU implements the Security operating state filtering. Otherwise, the GIC PMU does not use the Security operating state filtering. See *7.5 Event filtering* for more information.

No other Reusable event filter definitions are used.

$R_{JZTKK}$ It is IMPLEMENTATION DEFINED whether GIC PMU implements support for the Snapshot extension defined in [10].

$I_{ZRJLK}$ If the Snapshot extension is implemented, PMCFGR.SS is 1.

$R_{XTRZB}$ It is IMPLEMENTATION DEFINED whether the GIC PMU does implements the Trace generation extension defined in [10].

$R_{RRYKG}$ A GIC PMU does not implement the Export extension defined in [10]. For GIC PMU, PMCFG.EX is 0.

$R_{GYDWV}$ A GIC PMU does not implement the Dual-page extension defined in [10].

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 7.2 GIC PMU Overflow interrupt

$R_{HVTMV}$ A GIC PMU implements an overflow interrupt.

The permitted overflow interrupt types depend on which type of agent is being monitored as follows:

- If the GIC PMU counts events for an IRS, or for an IRS and its associated ITSs, the overflow interrupt is an SPI connected to that IRS.
- If the GIC PMU counts events for a single ITS, the overflow interrupt is one of the following:
  - An interrupt signal connected to an SPI on the associated IRS.
  - An interrupt wire signal connected to an IWB.
  - An ITS event generated from the ITS PMU as a system peripheral and translated by the ITS.

$R_{DRWTL}$ If the overflow interrupt is an ITS event generated from the GIC PMU as a system peripheral and translated by the ITS, all of the following are true:

- The GIC PMU uses the message-signaled interrupt functionality to configure the overflow interrupt as defined in [10].
- PMCFGR.MSI is 1.
- PMIRQCR0 is IGNORED and access to the register is permitted to be RAZ/WI.
- The DeviceID for the ITS event is IMPLEMENTATION DEFINED.
- The EventID for the ITS event is programmed using PMIRQCR1.DATA.
- The originating Interrupt Domain is the Interrupt Domain corresponding to the physical address space configured for the message-signaled interrupt.

$I_{LLFNW}$ The *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10] defines rules and controls that specify the physical address space used for message-signaled interrupts.

These rules are extended to also support the Root and Realm physical address space for message-signaled interrupts. See 7.1 *CoreSight PMU extensions* for more information.

$R_{PTFKG}$ When Non-secure register access is enabled, message-signaled interrupts are always Non-secure.

---

**Note**

This replaces an incorrect statement in [10] that states the opposite rule for Non-secure register access and the MSI physical address space.

---

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 7.3 GIC PMU event types

$D_{QTVKY}$ GIC PMU events are identified by a 4-bit PMEVTYPE value specifying an *event type*, and a 12-bit PMEVTID value specifying the *event* for the event type.

The PMEVTYPE space is interpreted as follows:

- 0b0000: Architected IRS event.
- 0b0001: IMPLEMENTATION DEFINED IRS event.
- 0b0010: Architected ITS event.
- 0b0011: IMPLEMENTATION DEFINED ITS event.

All PMEVTYPE values not defined above are reserved.

See 7.6 *IRS PMU events* and 7.7 *ITS PMU events* for information about the architected events.

$I_{RWFQW}$ An implementation may implement support for fewer than 16 bits of PMEVTID. In this case, unimplemented upper bits are RES0.

$I_{VSWDJ}$ When an event is selected using GIC_PMEVTYPER<n>.{PMEVTYPE, PMEVTID}, GIC_PMEVTYPER<n>.V reports whether an implemented event is selected.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 7.4 Ordering and synchronization requirements for the GIC PMU

$R_{HYDJQ}$ The GIC PMU is permitted to count in scope events that occur after a Register Write Effect to PMCNTEN that enables the corresponding monitor.

The GIC PMU is required to count all in scope events that occur after a Register Read Effect to PMCNTEN that reads PMCNTEN.P<n> as 1.

The GIC PMU is permitted to count in scope events that occur after a Register Write Effect to to PMCNTEN that disables the corresponding monitor.

The GIC PMU is not permitted to count any events that occur after a Register Read Effect to PMCNTEN that reads PMCNTEN.P<n> as 0.

---

**Note**

A Register Write Effect to PMCNTEN can occur either as a write to PMCNTEN or as a write to PMCNTENSET or PMCNTENCLR. A Register Read Effect to PMCNTEN can occur either as a read of PMCNTEN or as a read of PMCNTENSET or PMCNTENCLR. See the *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10] for more information.

---

$R_{CZMDL}$ If the agent being monitored contains an IRS, and an *IRS synchronization request* is made, if the GIC PMU is configured to count events related to interrupt events that are synchronized by an IRS synchronization request, the change to the GIC PMU count is required to be observable by a read of PMEVCNTR<n> when the IRS synchronization request is complete.

$R_{LQGKM}$ If the agent being monitored contains an ITS, and an *ITS synchronization request* is made, if the GIC PMU is configured to count events related to ITS events that are synchronized by an ITS synchronization request, the change to the GIC PMU count is required to be observable by a read of PMEVCNTR<n> when the ITS synchronization request is complete.

See also:

- 4.5 *IRS synchronization requests*
- 5.2.4 *ITS synchronization requests*

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 7.5 Event filtering

$I_{JLHGK}$ The GIC PMU supports filtering of events using various filters. The available filters depend on the event type and the event. See 7.6 *IRS PMU events* and 7.7 *ITS PMU events* for information about which filters are supported for the event types and architected events.

$I_{NYZYT}$ When an event is selected using GIC_PMEVTYPER<n>.{PMEVTYPE, PMEVTID}, GIC_PMEVTYPER<n>.FS reports whether filtering is supported for the selected event.

$R_{MZLWX}$ When more than one filter for an event is enabled, the event is counted if and only if it matches on all enabled filters.

$I_{RFZHB}$ When a GIC PMU implements the observability and access control extension, the GIC PMU supports filtering of events based on their association with the Security state using the Security operating state filtering defined as part of the Reusable event filter definitions defined in *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10].

The Security operating state filtering State match controls are implemented using GIC_PMEVTYPER<n>.{RL,EL3,NS,S}.

By filtering events using the Security state, the GIC PMU filters events matching the corresponding Interrupt Domains.

$D_{DGSKD}$ On some filters, the GIC PMU supports *filtering on a range* of values instead of matching on an exact value.

When filtering on a range, the range is specified by splitting the value into X most significant bits that must match, and Y least significant bits that are allowed to differ as follows:

- Bit[Y-1] is 0. That is, the most significant bit of the group of bits that do not need to match is zero.
- Bits[(Y-2:0] are 1. That is, the remaining bits that are allowed to differ are all programmed to 1.
- The remainder of bits (the X most-significant bits from bit Y upwards) contain a value that must match the corresponding bits for the value associated with monitored event.

$I_{GSLNM}$ GIC_PMEVTYPER<n> FSPAN reports whether filtering by a range of values is supported for the selected event.

$R_{MZTYJ}$ For each event that may support filtering on a range of values, it is IMPLEMENTATION DEFINED whether filtering on a range of values is supported or filtering is only supported on an exact value.

$S_{LKPXH}$ Arm expects that when a virtual GIC PMU is presented to a VM, the virtual GIC PMU does not support filtering on a range of values for identifiers where ranges in the host system do not correspond to similar ranges in the guest system.

$I_{VMFDS}$ The following are examples (in binary) of specifying ranges for GIC PMU filtering:

- `0000:0000:0001:1011:1111:0111:1111:0111` matches `0000:0000:0001:1011:1111:0111:1111:xxxx`
- `0000:0000:0001:1011:1111:0111:1111:0110` matches `0000:0000:0001:1011:1111:0111:1111:011x`
- `0000:0000:0001:1011:1111:0101:1111:1111` matches `0000:0000:0001:1011:1111:01xx:xxxx:xxxx`

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 7.6 IRS PMU events

RJPYSX The architected IRS PMU events are listed in Table 7.3.

**Table 7.3: Architected IRS PMU events**

| PMEVTYPE | PMEVTID | Name | Description |
|---|---|---|---|
| 0 | 0 | PMIRSCC | Counts IRS cycles. |
| 0 | 1 | PMIRSPENDSETIN | Counts incoming interrupt events to make an interrupt Pending. The events are counted when received from an SPI, an ITS, or another IRS. |
| 0 | 2 | PMIRSPENDCLEARIN | Counts incoming interrupt events to make an interrupt Idle. The events are counted when received from an SPI, an ITS, or another IRS. |
| 0 | 3 | PMIRSPENDSETOUT | Counts incoming interrupt events to make an interrupt Pending that are forwarded to another IRS. |
| 0 | 4 | PMIRSPENDCLROUT | Counts incoming interrupt events to make an interrupt Idle that are forwarded to another IRS. |
| 0 | 5 | PMIRSPENDSETPRO | Counts interrupt events to make an interrupt Pending that are processed at this IRS. |
| 0 | 6 | PMIRSPENDCLRPRO | Counts interrupt events to make an interrupt Idle that are processed at this IRS. |
| 0 | 7 | PMIRSPECMDSEND | Counts commands received from a PE to update an interrupt Pending state. |
| 0 | 8 | PMIRSPECMDHANDLE | Counts commands received from a PE related to handling of an interrupt. Includes only Activate and Deactivate commands. |
| 0 | 9 | PMIRSPECMDCFG | Counts commands received from a PE to update or request the configuration and state of an interrupt. |
| 0 | 10 | PMIRSVPEMAPPER | Counts when a VPE is made resident on an IRS and was not previously resident on that IRS. |
| 0 | 11 | PMIRSISTMISS | IST cache miss. Counts for each interrupt effect requiring an access to the IST. |
| 0 | 12 | PMIRSVMTMISS | VM table cache miss. Counts for each interrupt effect to a virtual interrupt requiring an access to the VM table. |
| 0 | 13 | PMIRSVPETMISS | VPE table cache miss. Counts for each interrupt effect to a virtual interrupt requiring an access to the VPE table. |
| 0 | 14 | PMIRSVMDMISS | VM descriptor miss. Counts for each interrupt effect to a virtual interrupt requiring an access to the VM descriptor. |
| 0 | 15 | PMIRSVPEDMISS | VPE descriptor miss. Counts for each interrupt effect to a virtual interrupt requiring an access to the VPE descriptor. |
| 0 | 16 | PMIRSL1ISTLU | Level 1 IST table access. Counts for each memory access to a level 1 IST. |
| 0 | 17 | PMIRSL2ISTLU | Level 2 IST table access. Counts for each memory access to a level 2 IST. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| PMEVTYPE | PMEVTID | Name | Description |
|---|---|---|---|
| 0 | 18 | PMIRSL1VMTLU | Level 1 VM table access. Counts for each memory access to a level 1 VM table. |
| 0 | 19 | PMIRSL2VMTLU | Level 2 VM table access. Counts for each memory access to a level 2 VM table. |
| 0 | 20 | PMIRSVPETLU | VPE table access. Counts for each memory access to a VPE table. |
| 0 | 21 | PMIRSVMDLU | VM descriptor access. Counts for each memory access to a VM descriptor. |
| 0 | 22 | PMIRSVPEDLU | VPE descriptor access. Counts for each memory access to a VPE descriptor. |
| 0 | 23 | PMIRSVPEDB | Counts when a VPE doorbell event is generated. |
| 0 | 24 | PMIRS1NPESEL | Counts when a new PE VPE selected for a 1ofN interrupt. |
| 0 | 25 | PMIRS1NDB | Counts when the 1ofN doorbell configuration for a VM causes a VPE doorbell to be generated. |

All other PMEVTID values for the architected IRS event type are reserved.

$R_{GFFRF}$ In a multi-IRS system, the events are counted only for the IRS being being monitored by the GIC PMU.

$S_{ZKMFB}$ Software is expected to combine events from multiple GIC PMUs to gather counts of events globally for the system.

$R_{CTPKW}$ When the IRS cycle counter event PMIRSCC is monitored, the event always counts IRS cycles, unless the IRS is in a low power state.

$R_{HBFMK}$ When an incoming interrupt event is received at the same IRS that processed the event, the event is counted both as incoming and as processing for the same interrupt event.

$R_{KKJVF}$ Table 7.4 lists the GIC System instructions that are processed by the IRS and recorded by IRS PMU events when the PE indicates to the IRS that the instruction is being executed. See 2.6 *GIC System instructions* for more information.

**Table 7.4: IRS PMU events counting processing of GIC System instructions**

| PMIRSPECMDPEND | PMIRSPECMDHANDLE | PMIRSPECMDCFG |
|---|---|---|
| `GIC xDPEND` | `GICR CDIA` | `GIC xDDIS` |
| | `GICR CDNMIA` | `GIC xDEN` |
| | `GIC xDDI` | `GIC xDRCFG` |
| | | `GIC xDPRI` |
| | | `GIC xDAFF` |
| | | `GIC xDHM` |

$R_{XJKZC}$ When monitoring any of the following events, the event is counted when the IRS performs any access to IRS data structures in memory as part of processing an Interrupt Effect or when making a VPE resident:

- 11: PMIRSISTMISS.
- 12: PMIRSVMTMISS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 13: PMIRSVPETMISS.
- 14: PMIRSVMDMISS.
- 15: PMIRSVPEDMISS.

$R_{JJNGJ}$ The IRS data structure access events are counted for an access to the corresponding IRS data structure in memory for any of the following reasons:

- There was an access to the data structure related to an Interrupt Effect.
- There was an access to the data structure related to making a VPE resident.
- There was a speculative access performed by the IRS.

### 7.6.1 IRS PMU events filtering

$R_{CSDPD}$ The GIC PMU supports the following filter types for the IRS events:

- Filtering on INTID.
- Filtering on physical or virtual interrupts, and a specific VM ID for virtual interrupts (VM and VM ID).
- Filtering on whether an access to an IRS data structure was a read or a write (RW).
- Filtering on the which PE sent a PE command to the IRS (Source PE).

Table 7.5 shows which filters are supported for the architected IRS events.

**Table 7.5: Supported filters for IRS PMU events**

| PMEVTID | Name | INTID filter | VM and VM ID filter | RW filter | Source PE filter |
|---|---|---|---|---|---|
| 0 | PMIRSCC | No | No | No | No |
| 1 | PMIRSPENDSETPRIN | Yes | Yes | No | No |
| 2 | PMIRSPENDCLRPRIN | Yes | Yes | No | No |
| 3 | PMIRSPENDSETOUT | Yes | Yes | No | No |
| 4 | PMIRSPENDCLROUT | Yes | Yes | No | No |
| 5 | PMIRSPENDSETPRO | Yes | Yes | No | No |
| 6 | PMIRSPENDCLRPRO | Yes | Yes | No | No |
| 7 | PMIRSPEVMPEND | Yes | Yes | No | Yes |
| 8 | PMIRSPECMDHANDLE | Yes | Yes | No | Yes |
| 9 | PMIRSPECMDCFG | Yes | Yes | No | Yes |
| 10 | PMIRSVPEMIGR | No | Yes[a] | No | Yes |
| 11 | PMIRSISTMISS | Yes | Yes | Yes | No |
| 12 | PMIRSVMTMISS | No | Yes | Yes | No |
| 13 | PMIRSVPETMISS | No | Yes | Yes | No |
| 14 | PMIRSVMDMISS | No | Yes | Yes | No |
| 15 | PMIRSVPEDMISS | No | Yes | Yes | No |
| 16 | PMIRSL1ISTLU | Yes | Yes | Yes | No |
| 17 | PMIRSL2ISTLU | Yes | Yes | Yes | No |
| 18 | PMIRSL1VMTLU | No | Yes | Yes | No |
| 19 | PMIRSL2VMTLU | No | Yes | Yes | No |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| PMEVTID | Name | INTID filter | VM and VM ID filter | RW filter | Source PE filter |
|---|---|---|---|---|---|
| 20 | PMIRSVPETLU | No | Yes | Yes | No |
| 21 | PMIRSVMDLU | No | Yes | Yes | No |
| 22 | PMIRSVPEDLU | No | Yes | Yes | No |
| 23 | PMIRSVPEDB | No | Yes | No | No |
| 24 | PMIRS1NPESEL | Yes | Yes | No | No |
| 25 | PMIRS1NDB | No | Yes | No | No |

a. The event is not counted when filtering on physical events only.

All other PMEVTID values for the architected IRS event type are reserved.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 7.7 ITS PMU events

$R_{GCNFY}$ The architected ITS PMU events are listed in Table 7.6.

**Table 7.6: Architected ITS PMU events**

| PMEVTYPE | PMEVTID | Name | Description |
|---|---|---|---|
| 2 | 0 | PMITSCC | Counts ITS cycles. |
| 2 | 1 | PMITSTRQ | Counts for each translation request performed by the ITS. |
| 2 | 2 | PMITSDIDMISS | DeviceID cache miss. Counts for each translation of a DeviceID that could not be satisfied from cache. |
| 2 | 3 | PMITSEIDMISS | EventID cache miss. Counts for each translation of an EventID that could not be satisfied from cache. |
| 2 | 4 | PMITSL1DTLU | L1 device table lookup. Counts every time a level 1 DTE is looked up in memory. Does not count when the entry is read from cache. |
| 2 | 5 | PMITSL2DTLU | L2 device table lookup. Counts every time a level 2 DTE is looked up in memory. Does not count when the entry is read from cache. |
| 2 | 6 | PMITSL1ITTLU | L1 interrupt translation table lookup. Counts every time a level 1 ITTE is looked up in memory. Does not count when the entry is read from cache. |
| 2 | 7 | PMITSL2ITTLU | L2 interrupt translation table lookup. Counts every time a level 2 ITTE is looked up in memory. Does not count when the entry is read from cache. |

All other PMEVTYPE values for the architected ITS event type are reserved.

$R_{MSFLT}$ When the ITS cycle counter event PMITSCC is monitored, it counts regardless of the event being translated by the ITS, unless the ITS is in the low power state.

$R_{YFQCW}$ The ITS DeviceID and EventID cache miss events, PMITSDIDMISS and PMITSEIDMISS, are counted when the translation causes any access to the ITS translation structures in memory to perform the translation.

$R_{ZJTPS}$ The ITS table lookup events are counted for each access to an ITS translation structure in memory caused by performing a translation or by speculative accesses.

$R_{GKBVK}$ ITS events are counted regardless of whether the translation was successful or not.

### 7.7.1 ITS PMU events filtering

$R_{HXGVQ}$ A GIC PMU supports filtering of ITS events by any of the following combinations of DeviceID and EventID:

- No filtering on DeviceID or EventID.
- DeviceID.
- DeviceID and EventID.

$R_{LNVGM}$ Filtering on DeviceID and EventID is supported for all other ITS events than ITS event 0 PMITSCC.

$I_{ZJXHT}$ When a GIC PMU counts events for an IRS and multiple associated ITSs, a monitor always counts ITS events for a single ITS specified by GIC_PMEVFILT2R<n>.ITSID.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{XGBDQ}$ When GIC_PMEVFILT2R<n>.FILTER_DID is 0, the monitor counts the selected PMU event for all translation requests.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1 and GIC_PMEVFILT2R<n>.FILTER_DID_SPAN is 0, the monitor counts the selected PMU event for translation requests where the DeviceID matches the DeviceID specified in GIC_PMEVFILTR<n>.DEVICE_ID.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1 and GIC_PMEVFILT2R<n>.FILTER_DID_SPAN is 1, the monitor counts the selected PMU event for translation requests where the DeviceID matches a range of DeviceIDs specified in GIC_PMEVFILTR<n>.DEVICE_ID.

If the ITS being monitored supports less than 2 bits of DeviceID, GIC_PMEVFILT2R<n>.FILTER_DID_SPAN is RES0 and matching a range of DeviceIDs is not supported.

$I_{NTCKY}$ When GIC_PMEVFILT2R<n>.FILTER_DID is 0, the monitor counts the selected PMU event for all translation requests, and filtering on EventID is not supported.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1 and GIC_PMEVFILT2R<n>.FILTER_EID is 0, the monitor counts the selected PMU event for translation requests where the DeviceID matches the DeviceID filter value, for all EventIDs.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1, GIC_PMEVFILT2R<n>.FILTER_EID is 1, and GIC_PMEVFILT2R<n>.FILTER_EID_SPAN is 0, the monitor counts the selected PMU event for translation requests where the DeviceID matches the DeviceID filter value, and the EventID matches the EventID specified in GIC_PMEVFILTR<n>.EVENT_ID.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1, GIC_PMEVFILT2R<n>.FILTER_EID is 1, and GIC_PMEVFILT2R<n>.FILTER_EID_SPAN is 1, the monitor counts the selected PMU event for translation requests where the DeviceID matches the DeviceID filter value, and the EventID matches the EventID matches a range of EventIDs specified in GIC_PMEVFILTR<n>.EVENT_ID.

If the ITS supports less than 2 bits of EventID, GIC_PMEVFILT2R<n>.FILTER_EID_SPAN is RES0 and matching a range of EventIDs is not supported.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 8
# System instructions

This chapter describes the CHCv5 system instructions.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 8.1 System instructions for the Current Interrupt Domain

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 8.1.1 GIC CDAFF, Interrupt Set Target in the Current Interrupt Domain

The GIC CDAFF characteristics are:

**Purpose**

Sets the routing mode and target PE for the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain, this operation applies to the virtual INTID in the VM identified by the resident VPE.

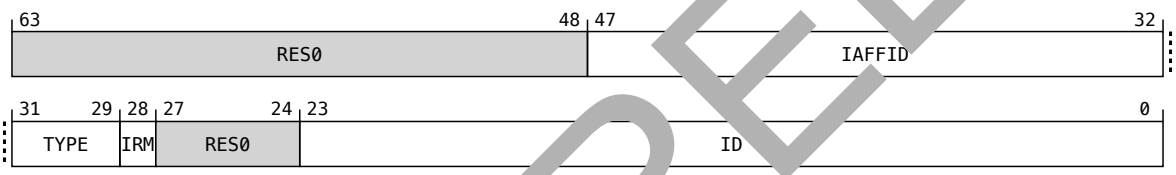
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDAFF are UNDEFINED.

**Attributes**

GIC CDAFF is a 64-bit System instruction.

## Field descriptions




**Bits [63:48]**

Reserved, RES0.

**IAFFID, bits [47:32]**

The interrupt Affinity value.

The IRS may support fewer than 16 bits of IAFFID. Unimplemented upper bits are treated as RES0 by the IRS.

When IRM is 1, this field provides an IMPLEMENTATION DEFINED hint to 1ofN selection algorithm. A value of 0 indicates that no hint is provided.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**IRM, bit [28]**

Controls how the interrupt is routed.

If the GIC IRS does not support 1ofN distribution of the specified interrupt Type, this configuration is treated as RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted |
| 0b1 | The interrupt Routing mode is 1ofN |

**Bits [27:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC CDAFF

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDAFF, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b011 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
       ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDAFF == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_AFF);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_AFF);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_AFF);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_AFF);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.2 GIC CDDI, Interrupt Deactivate in the Current Interrupt Domain

The GIC CDDI characteristics are:

**Purpose**

Clears the Active state of the specified INTID at the CPU interface in the Current Interrupt Domain.

This instruction applies to all interrupt types, including PPIs.

If the Current Interrupt Domain is the Virtual Interrupt Domain and the interrupt type is a PPI, this operation applies to the virtual PPI identified by the specified INTID.

If the Current Interrupt Domain is the Virtual Interrupt Domain and the interrupt type is not a PPI, this operation applies to the virtual INTID in the VM identified by the resident VPE.

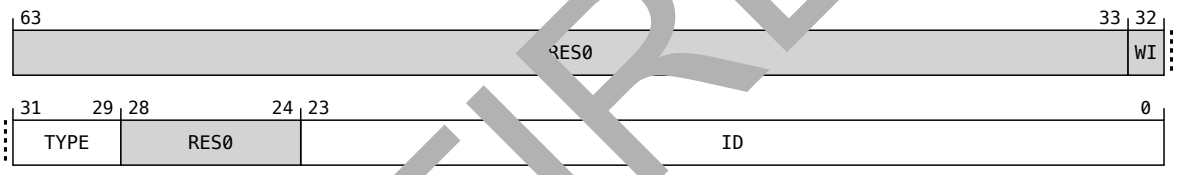
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDDI are UNDEFINED.

**Attributes**

GIC CDDI is a 64-bit System instruction.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**Bit [32]**

Reserved, WI.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

### Executing GIC CDDI

This instruction is treated as a NOP when any of the following are true:

- The INTID is unreachable.
- All of the following are true:
  - The interrupt type is not a PPI.
  - The operation applies to the Virtual Interrupt Domain.
  - There is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDDI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0010 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && !IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDDI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_DI);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DI);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DI);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.3 GIC CDDIS, Interrupt Disable in the Current Interrupt Domain

The GIC CDDIS characteristics are:

**Purpose**

Disables the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain this operation applies to the virtual INTID in the VM identified by the resident VPE.

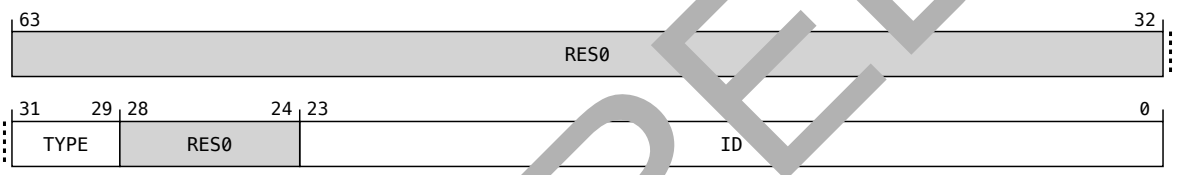
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDDIS are UNDEFINED.

**Attributes**

GIC CDDIS is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## Executing GIC CDDIS

This instruction has no effect if the INTID is unreachable

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDDIS, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDDIS == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_EL1, GICInstr_DIS);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DIS);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DIS);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_DIS);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.4 GIC CDEN, Interrupt Enable in the Current Interrupt Domain

The GIC CDEN characteristics are:

**Purpose**

Enables the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain, this operation applies to the virtual INTID in the VM identified by the resident VPE.

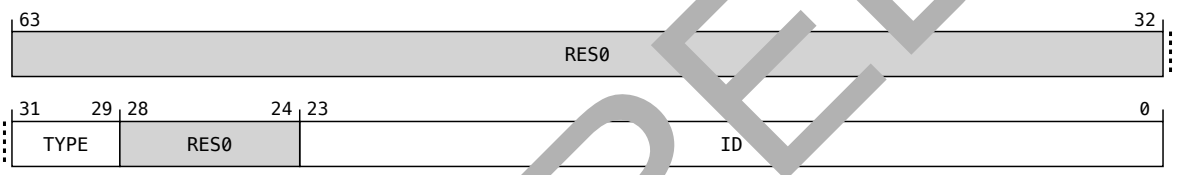
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDEN are UNDEFINED.

**Attributes**

GIC CDEN is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## Executing GIC CDEN

This instruction has no effect if the INTID is unreachable

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain, the interrupt type is not a PPI and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDEN, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_EN);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EN);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EN);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EN);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.5 GIC CDEOI, Priority Drop in the Current Interrupt Domain

The GIC CDEOI characteristics are:

**Purpose**

Performs a Priority Drop of the running priority at the CPU interface in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain, the Priority Drop applies to the virtual running priority.

Otherwise, it applies to the physical running priority in the Current Physical Interrupt Domain.

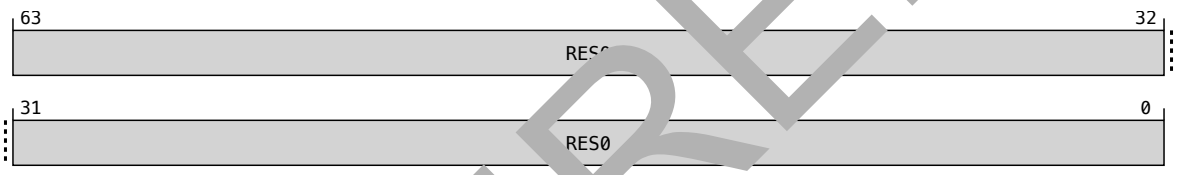
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDEOI are UNDEFINED.

**Attributes**

GIC CDEOI is a 64-bit System instruction.

#### Field descriptions




**Bits [63:0]**

Reserved, RES0.

#### Executing GIC CDEOI

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDEOI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b111 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDEOI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_EOI);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EOI);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EOI);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_EOI);
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 8.1.6 GIC CDHM, Interrupt Handling mode state in the Current Interrupt Domain

The GIC CDHM characteristics are:

**Purpose**

Sets the Handling mode of the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain this operation applies to the virtual INTID in the VM identified by the resident VPE.

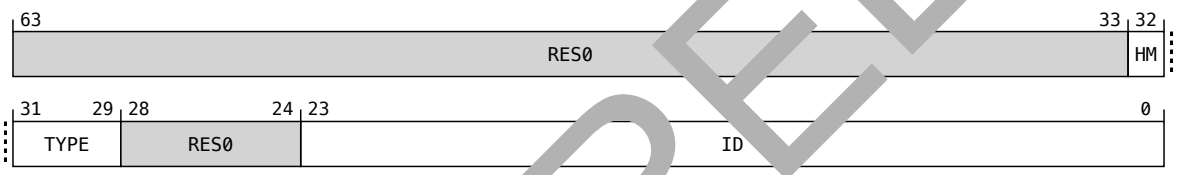
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDHM are UNDEFINED.

**Attributes**

GIC CDHM is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HM, bit [32]**

Handling mode

| HM | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | Level |

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC CDHM

This instruction has no effect if the INTID is unreachable

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDHM, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0010 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDHM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_HM);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_HM);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_HM);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_HM);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.7 GIC CDPEND, Interrupt Set/Clear Pending state in the Current Interrupt Domain

The GIC CDPEND characteristics are:

**Purpose**

Generates a SET or CLEAR event for the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain this operation applies to the virtual INTID in the VM identified by the resident VPE.

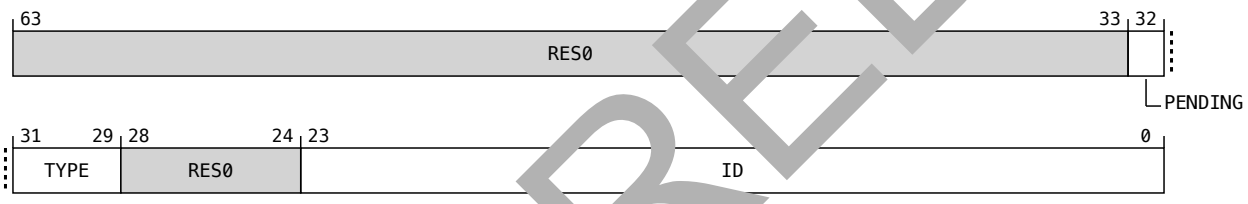
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDPEND are UNDEFINED.

**Attributes**

GIC CDPEND is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**PENDING, bit [32]**

Pending status

| PENDING | Meaning |
|---|---|
| 0b0 | Generate CLEAR event to the IRS. |
| 0b1 | Generate SET event to the IRS. |

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC CDPEND

This instruction has no effect if the INTID is unreachable

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDPEND, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b100 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDPEND == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_PEND);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PEND);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PEND);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PEND);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.8 GIC CDPRI, Interrupt Set priority in the Current Interrupt Domain

The GIC CDPRI characteristics are:

**Purpose**

Sets the priority for the specified INTID in the Current Interrupt Domain.

If the Current Interrupt Domain is the Virtual Interrupt Domain this operation applies to the virtual INTID in the VM identified by the resident VPE.

**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDPRI are UNDEFINED.

**Attributes**

GIC CDPRI is a 64-bit System instruction.

#### Field descriptions




**Bits [63:40]**

Reserved, RES0.

**PRIORITY, bits [39:35]**

The priority of the specified IN[illegible]D.

**Bits [34:32]**

Reserved, [illegible]S0.

**TYPE, bits [31:29]**

T[illegible]e of th[illegible]interrup

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## Executing GIC CDPRI

This instruction has no effect if the INTID is unreachable

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDPRI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b010 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GICCDPRI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_PRI);
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PRI);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PRI);
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_CD, GICInstr_PRI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.9 GIC CDRCFG, Request Interrupt Configuration in the Current Interrupt Domain

The GIC CDRCFG characteristics are:

**Purpose**

Request to read configuration of the specified INTID in the Current Interrupt Domain into ICC_ICSR_EL1.

If the Current Interrupt Domain is the Virtual Interrupt Domain, this operation applies to the virtual INTID in the VM identified by the resident VPE.

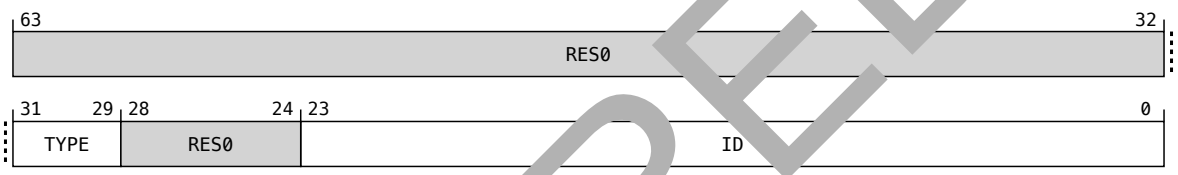
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GIC CDRCFG are UNDEFINED.

**Attributes**

GIC CDRCFG is a 64-bit System instruction.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## Executing GIC CDRCFG

This instruction is treated as a NOP if the operation applies to the Virtual Interrupt Domain and there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC CDRCFG, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0001 | 0b101 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeat  eImple  ted(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2.GIC RCF == '0  hen
        AArch64.SystemAccessTrap(EL2, 0x1 );
    elsif EL2Enabled() && HCR_EL2.IMO =  1' then
        AArch64.GIC(X[t, 64], GICInstrDo n_VD  CFG);
    else
        AArch64.GIC(X[t, 64], GICIns omain_  GICInstr_RCFG);
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GIC strDomain_  GICI tr_RCFG);
elsif PSTATE.EL == EL3 the
    AArch64.GIC(X[t, 64]  ICI rDomain_CD,  CInstr_RCFG);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.10 GICR CDIA, Interrupt Acknowledge in the Current Interrupt Domain

The GICR CDIA characteristics are:

**Purpose**

Acknowledges the HPPI in the Current Interrupt Domain.

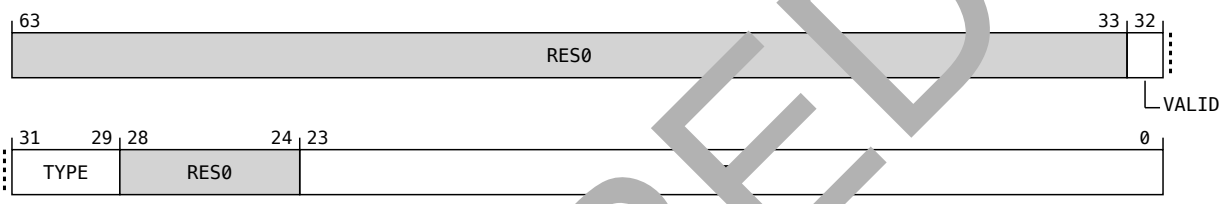
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GICR CDIA are UNDEFINED.

**Attributes**

GICR CDIA is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**VALID, bit [32]**

Indicates whether the instruction successfully acknowledged an interrupt.

| VALID | Meaning |
|---|---|
| 0b0 | No interrupt was acknowledged. |
| 0b1 | The HPPI was acknowledged. |

When this field is 1, the instruction acknowledges an HPPI of Sufficient priority that is not an NMI.

**TYPE, bits [31:29]**

The type of the acknowledged interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

If VALID is 0, this field is RES0.

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ID, bits [23:0]**

The ID of the acknowledged interrupt.

If VALID is 0, this field is RES0.

## Executing GICR CDIA

This system instruction is an alias of the SYSL instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GICR <Xt>, CDIA**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b00 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && ureIm mente FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2 ICRCDI then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM == '1' then
        X[t, 64] = AArch64.GICR ICIns Domain 0, GICInstr_IA);
    else
        X[t, 64] = AArch64. R(GICInstrDo in_CD, GICInstr_IA);
elsif PSTATE.EL == EL2 th
    X[t, 64] = AArch64.GICR(GICI rDomain_CD, GICInstr_IA);
elsif PSTATE.EL == EL then
    X[t, 64] = AAr 4.GICR(GICInstr ain_CD, GICInstr_IA);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.1.11 GICR CDNMIA, Non-maskable Interrupt Acknowledge in the Current Interrupt Domain

The GICR CDNMIA characteristics are:

**Purpose**

Acknowledges the HPPI, if it is an NMI, in the Current Interrupt Domain.

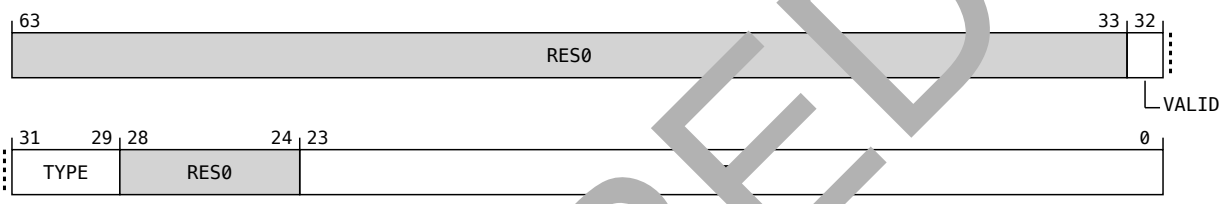
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GICR CDNMIA are UNDEFINED.

**Attributes**

GICR CDNMIA is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**VALID, bit [32]**

Indicates whether the instruction successfully acknowledged an NMI.

| VALID | Meaning |
|---|---|
| 0b0 | No interrupt was acknowledged. |
| 0b1 | The HPPI was acknowledged. |

When this field is 1, the instruction acknowledges an HPPI of Sufficient priority that is an NMI.

**TYPE, bits [31:29]**

The type of the acknowledged interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

If VALID is 0, this field is RES0.

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ID, bits [23:0]**

The ID of the acknowledged interrupt.

If VALID is 0, this field is RES0.

## Executing GICR CDNMIA

This system instruction is an alias of the SYSL instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GICR <Xt>, CDNMIA**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b00 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && [illegible]ureImp[illegible]mente[illegible]FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGITR_EL2[illegible]ICRCDN[illegible]2' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM[illegible]== '1'[illegible]en
        X[t, 64] = AArch64.GICR(GICInst[illegible]Domain[illegible]D, GICInstr_NMIA);
    else
        X[t, 64] = AArch64.[illegible]R(GICInstrDo[illegible]in_CD, GICInstr_NMIA);
elsif PSTATE.EL == EL2 th[illegible]
    X[t, 64] = AArch64.GICR(GICI[illegible]rDomain_CD, GICInstr_NMIA);
elsif PSTATE.EL == EL[illegible]then
    X[t, 64] = AArc[illegible]4.GICR(GICInstr[illegible]ain_CD, GICInstr_NMIA);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 8.2 System instructions for the Virtual Interrupt Domain

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 8.2.1 GIC VDAFF, Interrupt Set Target in the Virtual Interrupt Domain

The GIC VDAFF characteristics are:

**Purpose**

Sets the routing mode and target VPE for the specified INTID in the Virtual Interrupt Domain.

This operation applies to the virtual INTID in the VM identified by the resident VPE.

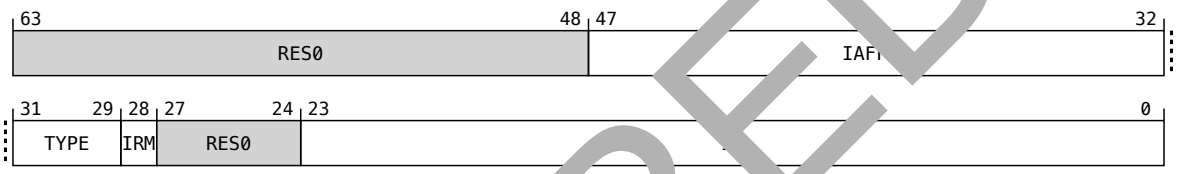
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDAFF are UNDEFINED.

**Attributes**

GIC VDAFF is a 64-bit System instruction.

#### Field descriptions




**Bits [63:48]**

Reserved, RES0.

**IAFFID, bits [47:32]**

The interrupt Affinity value.

The IRS may support fewer than 16 bits of IAFFID. Unimplemented upper bits are treated as RES0 by the IRS.

When IRM is 1, this field provides an IMPLEMENTATION DEFINED hint to 1ofN selection algorithm. A value of 0 means that no hint is provided.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**IRM, bit [28]**

Controls how the interrupt is routed.

If the GIC IRS does not support 1ofN distribution of the specified interrupt Type, this configuration is treated as RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted |
| 0b1 | The interrupt Routing mode is 1ofN |

**Bits [27:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC VDAFF

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if there is no resident VM.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encoding in the System instruction encoding space:

**GIC VDAFF, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0001 | 0b011 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_AFF);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_AFF);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.2 GIC VDDI, Interrupt Deactivate in the Virtual Interrupt Domain

The GIC VDDI characteristics are:

**Purpose**

Clears the Active state of the specified INTID in the Virtual Interrupt Domain.

This instruction applies to all interrupt types, including PPIs.

If the interrupt type is a PPI, this operation applies to the virtual PPI identified by the specified INTID.

If the interrupt type is not a PPI, this operation applies to the virtual INTID in the VM identified by the resident VPE.

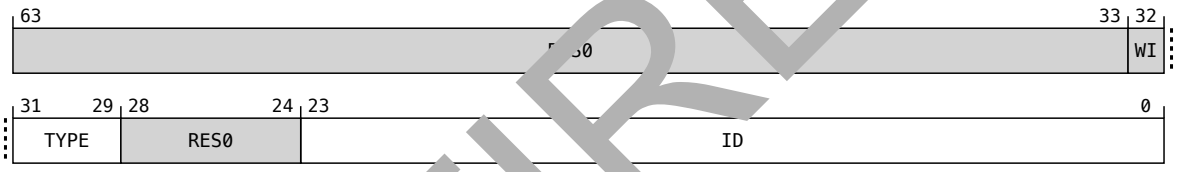
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDDI are UNDEFINED.

**Attributes**

GIC VDDI is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**Bit [32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

### Executing GIC VDDI

This instruction is treated as a NOP when any of the following are true:

- The INTID is unreachable.
- All of the following are true:
  - The interrupt type is not a PPI.
  - The operation applies to the Virtual Interrupt Domain.
  - There is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDDI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0010 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GICSysInstVector(t, 64[t], GICInstrDomain_VD, GICInstr_DI);
elsif PSTATE.EL == EL3 then
    if !IsGICEnabled() then
        UNDEFINED;
    else
        AArch64.GICSysInstVector(X[t, 64], GICInstrDomain_VD, GICInstr_DI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.3 GIC VDDIS, Interrupt Disable in the Virtual Interrupt Domain

The GIC VDDIS characteristics are:

**Purpose**

Disables the specified INTID in the Virtual Interrupt Domain.

This operation applies to the virtual INTID in the VM identified by the resident VPE.

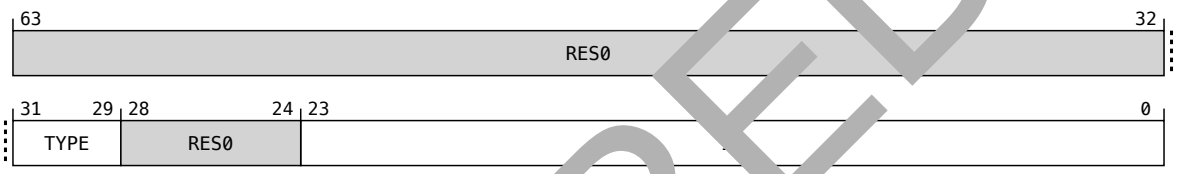
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDDIS are UNDEFINED.

**Attributes**

GIC VDDIS is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Value not defined above are reserved

**Bits [28:2**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

#### Executing GIC VDDIS

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if there is no resident VPE.

This system instruction is an alias of the SYS instruction.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDDIS, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0001 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDIS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDIS);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.4 GIC VDEN, Interrupt Enable in the Virtual Interrupt Domain

The GIC VDEN characteristics are:

**Purpose**

Enables the specified INTID in the Virtual Interrupt Domain.

This operation applies to the virtual INTID in the VM identified by the resident VPE.

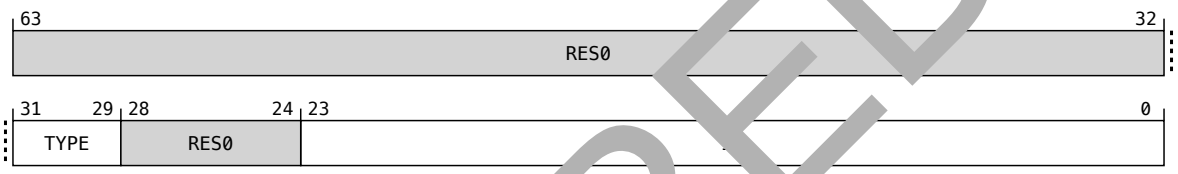
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDEN are UNDEFINED.

**Attributes**

GIC VDEN is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Value not defined above are reserved.

**Bits [28:2**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

#### Executing GIC VDEN

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if there is no resident VPE.

This system instruction is an alias of the SYS instruction.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDEN, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0001 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDEN);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDEN);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.5 GIC VDHM, Interrupt Handling mode in the Virtual Interrupt Domain

The GIC VDHM characteristics are:

**Purpose**

Sets the Handling mode of the specified INTID in the Virtual Interrupt Domain.

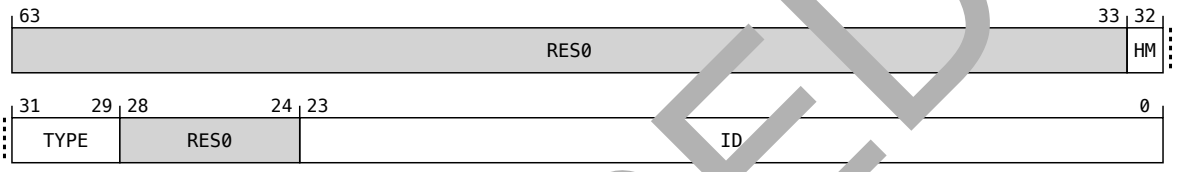
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDHM are UNDEFINED.

**Attributes**

GIC VDHM is a 64-bit System instruction.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HM, bit [32]**

Handling mode

| HM | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | Level |

**TYPE, bits [31:29]**

Type of

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

### Executing GIC VDHM

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDHM, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0[illegible] | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} [illegible](!IsFe[illegible]ureImpl[illegible]ed(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GC[illegible]_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x[illegible];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GIC[illegible]strDomain_[illegible] GICI[illegible]tr_HM);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GI[illegible]X[t, 64], GICInst[illegible]omain_VD, GICInstr_HM);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.6 GIC VDPEND, Interrupt Set/Clear Pending state in the Virtual Interrupt Domain

The GIC VDPEND characteristics are:

**Purpose**

Generates a SET or CLEAR event for the specified INTID in the Virtual Interrupt Domain.

This operation applies to the virtual INTID in the VM identified by the specified VM identifier.

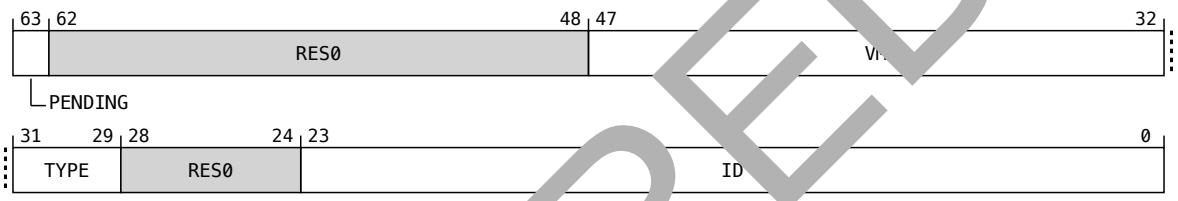
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDPEND are UNDEFINED.

**Attributes**

GIC VDPEND is a 64-bit System instruction.

#### Field descriptions




**PENDING, bit [63]**

Pending status

| PENDING | Meaning |
|---|---|
| 0b0 | Generate CLEAR event to the IRS. |
| 0b1 | Generate SET event to the IRS. |

**Bits [62:48]**

Reserved, RES0.

**VM, bits [47:32]**

The Virtual Machine identifier.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC VDPEND

This instruction has no effect if the INTID is unreachable.

This instruction has no effect if the VM is invalid.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDPEND, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b110 | 0b0001 | 0b100 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_PEND);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_PEND);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.7 GIC VDPRI, Interrupt Set priority in the Virtual Interrupt Domain

The GIC VDPRI characteristics are:

**Purpose**

Sets the priority for the specified INTID in the Virtual Interrupt Domain.

This operation applies to the virtual INTID in the VM identified by the resident VPE.

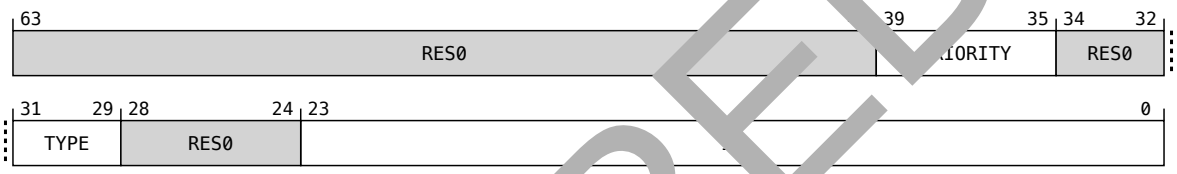
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDPRI are UNDEFINED.

**Attributes**

GIC VDPRI is a 64-bit System instruction.

#### Field descriptions




**Bits [63:40]**

Reserved, RES0.

**PRIORITY, bits [39:35]**

The priority of the specified INTID.

**Bits [34:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented lower bits are RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Executing GIC VDPRI

This instruction has no effect if the INTID is unreachable.

This instruction is treated as a NOP if there is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDPRI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0001 | 0b010 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VPE, GICInstr_PRI);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VPE, GICInstr_PRI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.2.8 GIC VDRCFG, Request Interrupt Configuration in the Virtual Interrupt Domain

The GIC VDRCFG characteristics are:

**Purpose**

Request to read configuration of the specified INTID in the Virtual Interrupt Domain into ICC_ICSR_EL1.

This operation applies to the virtual INTID in the VM identified by the resident VPE.

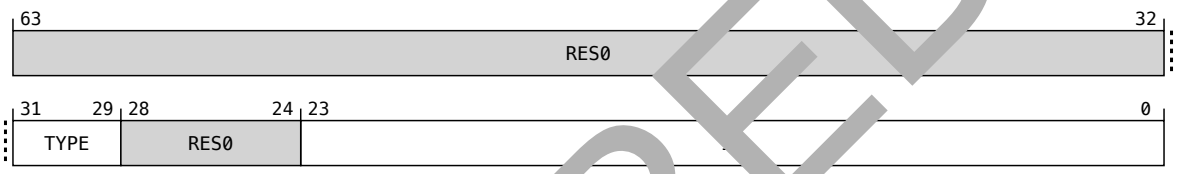
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to GIC VDRCFG are UNDEFINED.

**Attributes**

GIC VDRCFG is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Value not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

#### Executing GIC VDRCFG

This instruction is treated as a NOP if there is no resident VPE.

This system instruction is an alias of the SYS instruction.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC VDRCFG, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b100 | 0b1100 | 0b0001 | 0b101 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDRCFG);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.GIC(X[t, 64], GICInstrDomain_VD, GICInstr_VDRCFG);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 8.3 System instructions for the Logical Interrupt Domain

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 8.3.1 GIC LDAFF, Interrupt Set Target in the Logical Interrupt Domain

The GIC LDAFF characteristics are:

**Purpose**

Sets the routing mode and target PE for the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

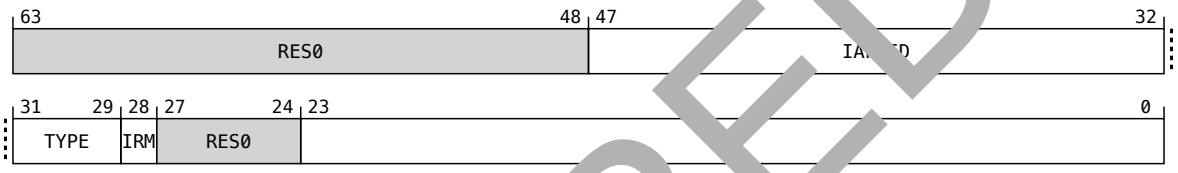
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDAFF are UNDEFINED.

**Attributes**

GIC LDAFF is a 64-bit System instruction.

## Field descriptions




**Bits [63:48]**

Reserved, RES0.

**IAFFID, bits [47:32]**

The interrupt Affinity v

The IRS may supp rt fewer tha 6 bits of IAFFID. Unimplemented upper bits are treated as RES0 by the IRS.

When IRM 1, this fi provides an IMPLEMENTATION DEFINED hint to 1ofN selection algorithm. A value of 0 means t no t is provided.

**TYPE, bi 29]**

ype of th interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**IRM, bit [28]**

Controls how the interrupt is routed.

If the GIC IRS does not support 1ofN distribution of the specified interrupt Type, this configuration is treated as RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted |
| 0b1 | The interrupt Routing mode is 1ofN |

**Bits [27:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC LDAFF

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having a value UNKNOWN value.
- The instruction is treated as a NOP.

This instruction has no effect if the INTID is unreachable

This system instruction is an alias of the GICS instruction.

Accesses to this instruction use the following encoding in the System instruction encoding space:

**GIC LDAFF, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b1 | 0b1100 | 0b0001 | 0b011 |

```
if !IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_AFF);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.2 GIC LDDI, Interrupt Deactivate in the Logical Interrupt Domain

The GIC LDDI characteristics are:

**Purpose**

Clears the Active state of the specified INTID in the Logical Interrupt Domain associated with the Security state selected by SCR_EL3.{NS, NSE}.

This instruction applies to all interrupt types, including PPIs.

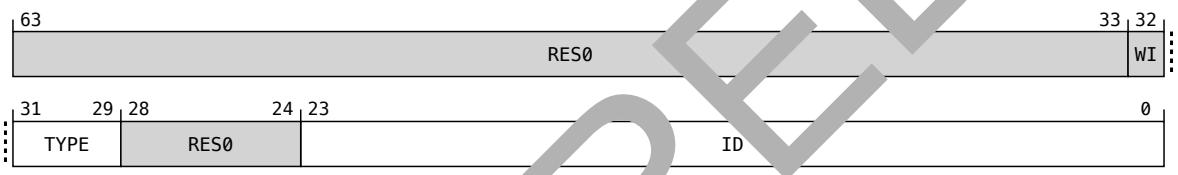
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDDI are UNDEFINED.

**Attributes**

GIC LDDI is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**Bit [32]**

Reserved, WI.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Executing GIC LDDI

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

This instruction is treated as a NOP when any of the following are true:

- The INTID is unreachable.
- All of the following are true:
  - The interrupt type is not a PPI.
  - The operation applies to the Virtual Interrupt Domain.
  - There is no resident VPE.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDDI, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0010 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_DI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.3 GIC LDDIS, Interrupt Disable in the Logical Interrupt Domain

The GIC LDDIS characteristics are:

**Purpose**

Disables the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

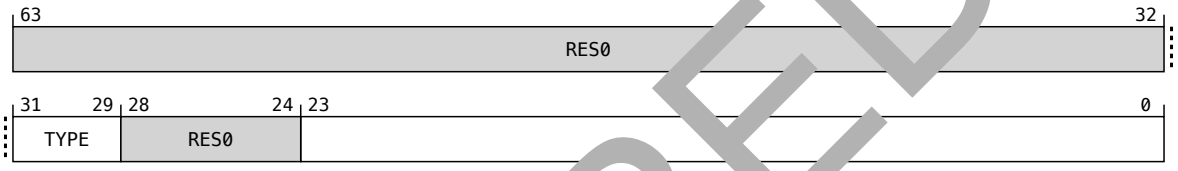
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDDIS are UNDEFINED.

**Attributes**

GIC LDDIS is a 64-bit System instruction.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Value[illegible] above are reserved

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC LDDIS

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

This instruction has no effect if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDDIS, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0001 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_DIS);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.4 GIC LDEN, Interrupt Enable in the Logical Interrupt Domain

The GIC LDEN characteristics are:

**Purpose**

Enables the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

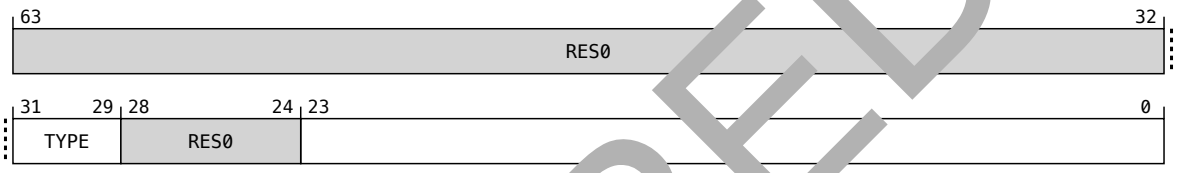
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDEN are UNDEFINED.

**Attributes**

GIC LDEN is a 64-bit System instruction.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC LDEN

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

This instruction has no effect if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDEN, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0001 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_LDEN);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.5 GIC LDHM, Interrupt Handling mode in the Logical Interrupt Domain

The GIC LDHM characteristics are:

**Purpose**

Sets the Handling mode of the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

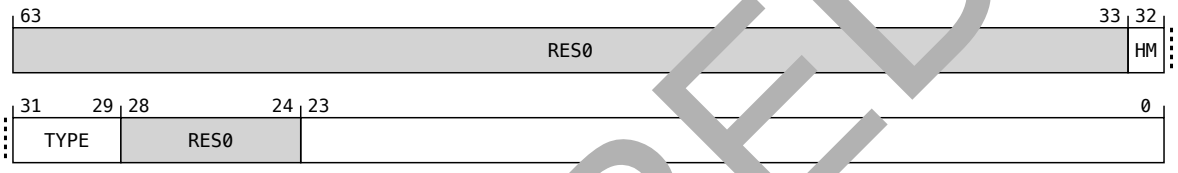
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDHM are UNDEFINED.

**Attributes**

GIC LDHM is a 64-bit System instruction.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HM, bit [32]**

Handling mode

| HM | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | Level |

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC LDHM

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

This instruction has no effect if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System inst encoding space:

**GIC LDHM, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0010 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstr_Domain_LD, GICInstr_HM);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.6 GIC LDPEND, Interrupt Set/Clear Pending state in the Logical Interrupt Domain

The GIC LDPEND characteristics are:

**Purpose**

Generates a SET or CLEAR event for the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

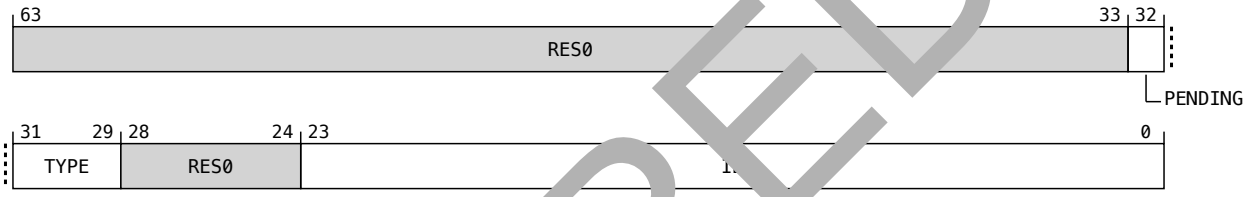
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDPEND are UNDEFINED.

**Attributes**

GIC LDPEND is a 64-bit System instruction.

#### Field descriptions




**Bits [63:33]**

Reserved, RES0.

**PENDING, bit [32]**

Pending status

| PENDING | Meaning |
|---|---|
| 0b0 | Generate CLEAR event to the IRS. |
| 0b1 | Generate SET event to the IRS. |

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

## Executing GIC LDPEND

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

This instruction has no effect if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDPEND, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0001 | 0b100 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_PEND);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.7 GIC LDPRI, Interrupt Set priority in the Logical Interrupt Domain

The GIC LDPRI characteristics are:

**Purpose**

Sets the priority for the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits.

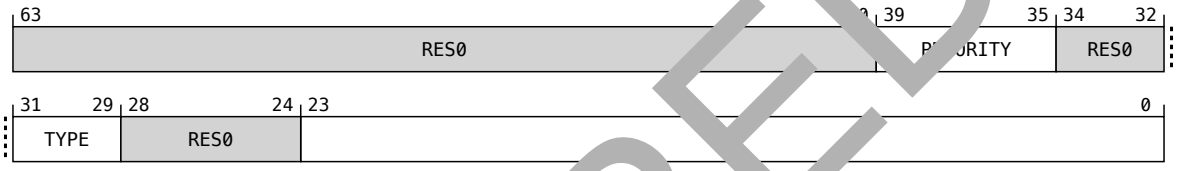
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDPRI are UNDEFINED.

**Attributes**

GIC LDPRI is a 64-bit System instruction.

## Field descriptions




**Bits [63:40]**

Reserved, RES0.

**PRIORITY, bits [39:35]**

The priority of the specified INTID.

**Bits [34:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### Executing GIC LDPRI

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

This instruction has no effect if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDPRI, <Xt>**

| op0 | op1 | CRn | CR | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0001 | 0b010 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICR_StrDomainCFGD, GICR_Str_PRI);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.3.8 GIC LDRCFG, Request Interrupt Configuration in the Logical Interrupt Domain

The GIC LDRCFG characteristics are:

**Purpose**

Request to read configuration of the specified INTID in the Logical Interrupt Domain associated with the Security state selected by the SCR_EL3.{NS, NSE} bits into ICC_ICSR_EL1.

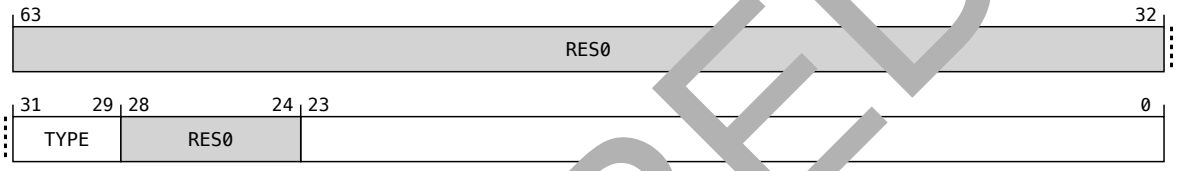
**Configuration**

This system instruction is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to GIC LDRCFG are UNDEFINED.

**Attributes**

GIC LDRCFG is a 64-bit System instruction.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**TYPE, bits [31:29]**

Type of the interrupt

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values [illegible] above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The ID of the interrupt.

ID and TYPE together form an INTID.

The number of ID bits implemented is reported in ICC_IDR0_EL1.ID_BITS. Unimplemented upper bits are RES0.

#### Executing GIC LDRCFG

When executed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The instruction is treated as a NOP.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

ICC_ICSR_EL1.F is set to 1 and other fields become UNKNOWN after execution of this instruction if the INTID is unreachable.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GIC LDRCFG, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b110 | 0b1100 | 0b0001 | 0b101 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.GIC(X[t, 64], GICInstrDomain_LD, GICInstr_RDLDCFG);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 8.4 GIC synchronization barrier instructions

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 8.4.1 GSB SYS, GIC Synchronization Barrier System

The GSB SYS characteristics are:

**Purpose**

GIC Synchronization Barrier System ensures completion of the effects of GIC instructions and prevents loads, stores, and GIC instructions from executing part of their functionality before the GSB SYS.

See 'Interrupt ordering model and synchronization requirements' in *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5* (ARM AES 0070) for more information.

**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GSB SYS are UNDEFINED.

**Attributes**

GSB SYS is a 64-bit System instruction.

#### Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

#### Executing GSB SYS

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GSB SYS**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    else
        AArch64.GSB(GICInstr_SYS);
elsif PSTATE.EL == EL2 then
    AArch64.GSB(GICInstr_SYS);
elsif PSTATE.EL == EL3 then
    AArch64.GSB(GICInstr_SYS);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 8.4.2 GSB ACK, GIC Synchronization Barrier Interrupt Acknowledge

The GSB ACK characteristics are:

**Purpose**

GIC Synchronization Barrier Interrupt Acknowledge ensures completion of the effects of the GICR CDIA and GICR CDNMIA instructions and prevents loads, stores, and GIC instructions from executing part of their functionality before the GSB ACK.

See 'Interrupt ordering model and synchronization requirements' in *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5* (ARM AES 0070) for more information.

**Configuration**

This system instruction is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to GSB ACK are UNDEFINED.

**Attributes**

GSB ACK is a 64-bit System instruction.

#### Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

#### Executing GSB ACK

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

**GSB ACK**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b01 | 0b000 | 0b1100 | 0b0000 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    else
        AArch64.GSB(GICInstr_ACK);
elsif PSTATE.EL == EL2 then
    AArch64.GSB(GICInstr_ACK);
elsif PSTATE.EL == EL3 then
    AArch64.GSB(GICInstr_ACK);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter 9
# System registers

This chapter describes the CHCv5 System registers.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.1 Synchronization requirements for GICv5 System registers

$I_{CWKLD}$ The GICv5 System registers follow the requirements for System registers in the *Arm® Architecture Reference Manual, for A-profile architecture*[1].

$R_{NWMPV}$ The effects of direct writes to any of the following System registers are visible to instructions in program order after the direct write:

- ICC_PCR_EL3
- ICC_PCR_EL1
- ICV_PCR_EL1

The effects of a direct write to any of the above System registers are the following:

- If an interrupt is no longer of sufficient priority it is not reported as the HPPI after a read from any of the following registers:
  - ICC_HPPIR_EL3.
  - ICC_HPPIR_EL1.
  - ICV_HPPIR_EL1.
  - ICC_DOMHPPIR_EL3
- If an interrupt is no longer of sufficient priority it cannot be acknowledged by a `GICR <Xt>, CDIA` or `GICR <Xt>, CDNMIA` System instruction.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 9.2 CPU interface registers

Configuration and state of the CPU interface.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.1 ICC_APR_EL1, Interrupt Controller Physical Active Priorities Register (EL1)

The ICC_APR_EL1 characteristics are:

**Purpose**

Records physical active priorities for the Non-secure, Realm, and Secure Interrupt Domains.

**Configuration**

There are separate banked copies of this register for Non-secure, Realm and Secure state.

This register is banked between ICC_APR_EL1 and ICC_APR_EL1_NS and ICC_APR_EL1_RL and ICC_APR_EL1_S.

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_APR_EL1 are UNDEFINED.

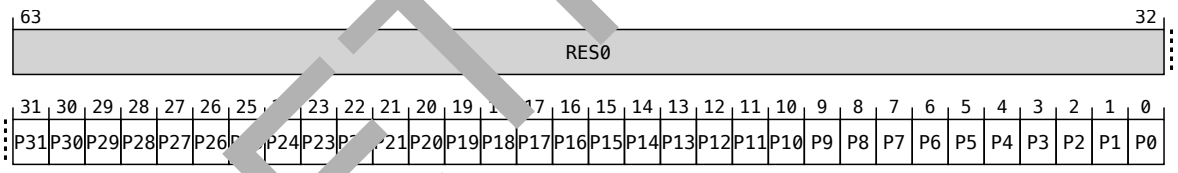
**Attributes**

ICC_APR_EL1 is a 64-bit register.

This register has the following instances:

- ICC_APR_EL1, when EL3 is not implemente
- ICC_APR_EL1_NS, when EL3 is implemented.
- ICC_APR_EL1_RL, when FEAT_R is im emente
- ICC_APR_EL1_S, when (H eEL(E & & (!IsFeatureImplemented(FEAT_RME))) || (IsFeatureImplemented(FEAT_RME) & sFeatureImplemented(FEAT_SEL2)).

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**P<x>, bits [x], for x = 31 to 0**

Provides access to the active priorities.

| P<x> | Meaning |
|---|---|
| 0b0 | Priority not active |
| 0b1 | Priority active |

Fields in this register are indexed using the 5-bit priority as an unsigned integer, P[Priority].

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Accessing ICC_APR_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_APR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_APR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_APR_EL1;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_APR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_APR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_APR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_APR_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_APR_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_APR_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_APR_EL1_NS;
    else
        UNDEFINED;
```

**MSR ICC_APR_EL1, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b000 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_APR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_APR_EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_APR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_APR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_APR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_APR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_APR_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        ICC_APR_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        ICC_APR_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.2 ICC_APR_EL3, Interrupt Controller Physical Active Priorities Register for EL3

The ICC_APR_EL3 characteristics are:

**Purpose**

Records active priorities for the EL3 Interrupt Domain.

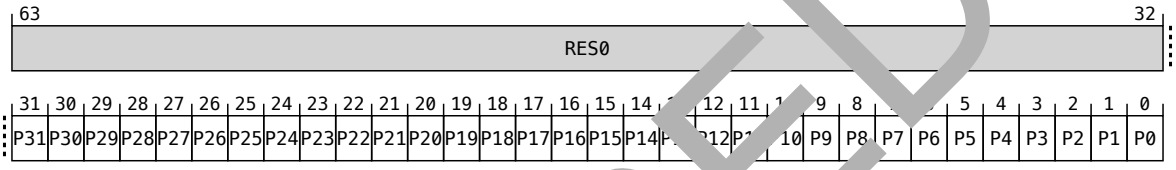
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_APR_EL3 are UNDEFINED.

**Attributes**

ICC_APR_EL3 is a 64-bit register.

#### Field descriptions




**Bits [63:32]**

Reserved, RES0.

**P<x> , bits [x], for x = 31 to 0**

Provides access to the acti priorities.

| P<x> | Meaning |
|---|---|
| 0b0 | Priority not active |
| 0b1 | Priority active |

ields in is register e indexed using the 5-bit priority as an unsigned integer, P[Priority].

t behavior of this field is:

- a Warm reset, this field resets to an UNKNOWN value.

#### Accessing ICC_APR_EL3

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_APR_EL3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1000 | 0b000 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_APR_EL3;
```

### MSR ICC_APR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b10 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) th
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_APR_EL3 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.3 ICC_CR0_EL1, Interrupt Controller Physical Control Register (EL1)

The ICC_CR0_EL1 characteristics are:

**Purpose**

Controls behavior of the physical CPU interface for the Non-secure, Realm, and Secure Interrupt Domains.

**Configuration**

There are separate banked copies of this register for Non-secure, Realm and Secure state.

When EL3 is implemented, AArch64 System register ICC_CR0_EL1 bits [2:1] are architecturally mapped to AArch64 System register ICC_CR0_EL3[2:1].

This register is banked between ICC_CR0_EL1 and ICC_CR0_EL1_NS and ICC_CR0_EL1_RL and ICC_CR0_EL1_S.

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_CR0_EL1 are UNDEFINED.

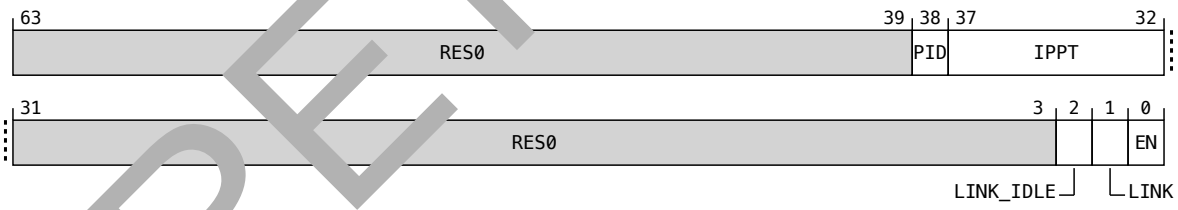
**Attributes**

ICC_CR0_EL1 is a 64-bit register.

This register has the following instances:

- ICC_CR0_EL1, when EL3 is not implemented.
- ICC_CR0_EL1_NS, when EL3 is implemented.
- ICC_CR0_EL1_RL, when FEAT_RME is implemented.
- ICC_CR0_EL1_S, when (HaveEL(EL3) && (!IsFeatureImplemented(FEAT_RME))) || (IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SEL2)).

## Field descriptions




**Bits [63:39]**

Reserved, RES0.

**PID, bit [38]**

**When EL3 is implemented:**

Preemptive Interrupt Domain. Indicates whether the Interrupt Domain associated with the Security state selected by SCR_EL3.{NSE,NS} is the Preemptive Interrupt Domain.

Access to this field is RO.

**Otherwise:**

Reserved, RAZ/WI.

**IPPT, bits [37:32]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When ICC_CR0_EL1.PID == 1:**

Interrupt Preemptive Priority Threshold value for the Preemptive Interrupt Domain.

---

**Note**

ICC_CR0_EL1.IPPT is a 6 bits field to ensure it can be strictly higher than the interrupt priority, which is a 5-bit unsigned value.

---

See 'Preemptive interrupts' in *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 5* (ARM AES 0070) for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [31:3]**

Reserved, RES0.

**LINK_IDLE, bit [2]**

Whether the link between the CPU interface and the IRI is in the process of connecting or disconnecting.

| LINK_IDLE | Meaning |
|---|---|
| 0b0 | The link between the CPU interface and the IRI is in the process of connecting or disconnecting. |
| 0b1 | The link between the CPU interface and the IRI is not in the process of connecting or disconnecting. |

When this field is 0, if the system has been properly configured and a connection can be made between the CPU interface and the IRI, the field will become 1 in finite time. If the system has not been properly configured and an attempt to connect the CPU interface to the IRI is made, this field may remain 0 indefinitely.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**LINK, bit [1]**

Whether the link between the CPU interface and the IRI is connected.

| LINK | Meaning |
|---|---|
| 0b0 | The link between the CPU interface and the IRI is disconnected. |
| 0b1 | The link between the CPU interface and the IRI is connected. |

On a read, all of the following are true:

- If LINK_IDLE is 1, LINK indicates whether the link is connected.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- If LINK_IDLE is 0 and LINK is 0, the link is in the process of disconnecting.
- If LINK_IDLE is 0 and LINK is 1, the link is in the process of connecting.

On a write, all of the following are true:

- Writing 1 connects the link.
- Writing 0 disconnects the link.
- The process of connecting or disconnecting the link is complete when LINK_IDLE is 1.
- A write that does not change the value of this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- Access to this field is RO if any of the following are true:
  - EL3 is implemented
  - ICC_CR0_EL1.LINK_IDLE == 0
- Otherwise, access to this field is RW.

**EN, bit [0]**

Enable interrupts for the Interrupt Domain.

When this field is 0, there is no HPPI of sufficient priority for the Interrupt Domain.

| EN | Meaning |
|---|---|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

## Accessing ICC_CR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_CR0_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_CR0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_CR0_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CR0_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_CR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CR0_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_CR0_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_CR0_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && S[illegible].NSE == '1' && [illegible]CR_EL3.NS == '1' then
        X[t, 64] = ICC_CR0_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.N[illegible] == '1' [illegible]en
        X[t, 64] = ICC_CR0_EL1_NS;
    else
        UNDEFINED;
```

## MSR ICC_CR0_EL1, <[illegible]c>

| op0 | op1 | [illegible]Rn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b[illegible] | 0b1100 | 0b0000 | 0b001 |

```
if !IsFe[illegible]pleme[illegible]d(FEA[illegible]GCIE) then
    UN[illegible]FINED;
elsi[illegible]PSTATE.[illegible] == EL0 t[illegible]n
    [illegible]DEFINE[illegible]
elsif [illegible]T[illegible].EL == EL[illegible] then
    if E[illegible]nabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        [illegible]H_VCTLR_EL2.V3 == '1' then
        UNDE[illegible]NED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_CR0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_CR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CR0_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_CR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            ICC_CR0_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_CR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_CR0_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        ICC_CR0_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        ICC_CR0_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.4 ICC_CR0_EL3, Interrupt Controller Physical Control Register (EL3)

The ICC_CR0_EL3 characteristics are:

**Purpose**

Controls behavior of the CPU interface in the EL3 Interrupt Domain.

**Configuration**

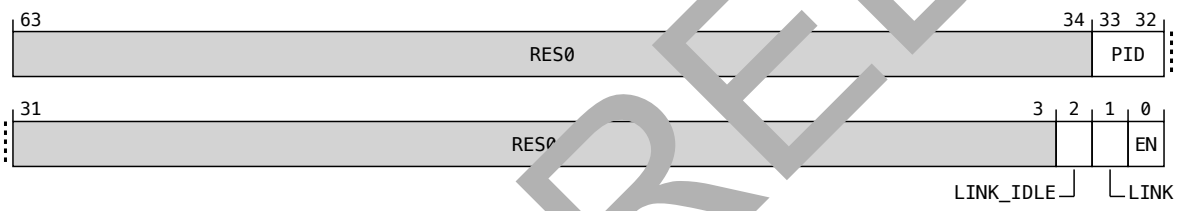
AArch64 System register ICC_CR0_EL3 bits [2:1] are architecturally mapped to AArch64 System register ICC_CR0_EL1[2:1].

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_CR0_EL3 are UNDEFINED.

**Attributes**

ICC_CR0_EL3 is a 64-bit register.

## Field descriptions




**Bits [63:34]**

Reserved, RES0.

**PID, bits [33:32]**

Preemptive Interrupt Domain.

The PE behaves as if there is no Preemptive Interrupt Domain when any of the following are true:

- This field is set to 0b10.
- This field is set to an Interrupt Domain that is not implemented.

| PID | Meaning |
|---|---|
| 0b00 | Secure Interrupt Domain |
| 0b01 | Non-secure Interrupt Domain |
| 0b10 | None |
| 0b11 | Realm Interrupt Domain |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [31:3]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**LINK_IDLE, bit [2]**

Whether the link between the CPU interface and the IRI is in the process of connecting or disconnecting.

| LINK_IDLE | Meaning |
|---|---|
| 0b0 | The link between the CPU interface and the IRI is in the process of connecting or disconnecting. |
| 0b1 | The link between the CPU interface and the IRI is not in the process of connecting or disconnecting. |

When this field is 0, if the system has been properly configured and a connection can be made between the CPU interface and the IRI, the field will become 1 in finite time. If the system has not been properly configured and an attempt to connect the CPU interface to the IRI is made, this field may remain 0 indefinitely.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Access to this field is RO.

**LINK, bit [1]**

Whether the link between the CPU interface and the IRI is connected.

| LINK | Meaning |
|---|---|
| 0b0 | The link between the CPU interface and the IRI is disconnected. |
| 0b1 | The link between the CPU interface and the IRI is connected. |

On a read, all of the following are true:

- If LINK_IDLE is 1, LINK indicates whether the link is connected.
- If LINK_IDLE is 0 and LINK is 0, the link is in the process of disconnecting.
- If LINK_IDLE is 0 and LINK is 1, the link is in the process of connecting.

On a write, all of the following are true:

- Writing 1 connects the link.
- Writing 0 disconnects the link.
- The process of connecting or disconnecting the link is complete when LINK_IDLE is 1.
- A write that does not change the value of this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When ICC_CR0_EL3.LINK_IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**EN, bit [0]**

Enable interrupts for the EL3 Interrupt Domain.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

When this field is set to 0, the PE behaves as if there is no HPPI of Sufficient priority in the EL3 Interrupt Domain.

| EN | Meaning |
|---|---|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

### Accessing ICC_CR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_CR0_EL3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1001 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_CR0_EL3;
```

**MSR ICC_CR0_EL3, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1001 | 0b000 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_CR0_EL3 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.5 ICC_DOMHPPIR_EL3, Interrupt Controller Domain Highest Priority Pending Interrupt Register

The ICC_DOMHPPIR_EL3 characteristics are:

**Purpose**

Reports the HPPI for a Non-EL3 Interrupt Domain.

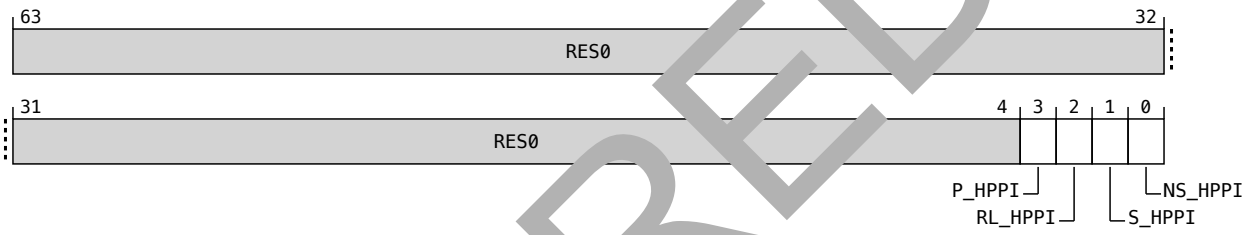
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_DOMHPPIR_EL3 are UNDEFINED.

**Attributes**

ICC_DOMHPPIR_EL3 is a 64-bit register.

## Field descriptions




**Bits [63:4]**

Reserved, RES0.

**P_HPPI, bit [3]**

Preemptive HPPI for the Preemptive Interrupt Domain valid bit.

| P_HPPI | Meaning |
|---|---|
| 0b0 | There is no Preemptive HPPI of Sufficient priority for the Preemptive Interrupt Domain. |
| 0b1 | There is an Preemptive HPPI of Sufficient priority for the Preemptive Interrupt Domain. |

Accessing this field has the following behavior:

- When HavePreemptiveDomain(), access to this field is RO.
- Otherwise, access to this field is RES0.

**RL_HPPI, bit [2]**

HPPI for Realm Interrupt Domain valid bit.

| RL_HPPI | Meaning |
|---|---|
| 0b0 | There is no HPPI of Sufficient priority for Realm Interrupt Domain. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| RL_HPPI | Meaning |
|---|---|
| 0b1 | There is an HPPI of Sufficient priority for Realm Interrupt Domain. |

Accessing this field has the following behavior:

- When HaveDomain(GICDomain_RL), access to this field is RO.
- Otherwise, access to this field is RES0.

**S_HPPI, bit [1]**

HPPI for Secure Interrupt Domain valid bit.

| S_HPPI | Meaning |
|---|---|
| 0b0 | There is no HPPI of Sufficient priority for Secure Interrupt Domain. |
| 0b1 | There is an HPPI of Sufficient priority for Secure Interrupt Domain. |

Accessing this field has the following behavior:

- When HaveDomain(GICDomain_S), access to this field is RO.
- Otherwise, access to this field is RES0.

**NS_HPPI, bit [0]**

HPPI for Non-secure Interrupt Domain valid bit.

| NS_HPPI | Meaning |
|---|---|
| 0b0 | There is no HPPI of Sufficient priority for Non-secure Interrupt Domain. |
| 0b1 | There is an HPPI of Sufficient priority for Non-secure Interrupt Domain. |

Accessing this field has the following behavior:

- When HaveDomain(GICDomain_NS), access to this field is RO.
- Otherwise, access to this field is RES0.

## Accessing ICC_DOMHPPIR_EL3

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_DOMHPPIR_EL3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1000 | 0b010 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_DOMHPPIR_EL3;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.6 ICC_HAPR_EL1, Interrupt Controller Physical Highest Active Priority Register

The ICC_HAPR_EL1 characteristics are:

**Purpose**

Reports the running priority of the Current Physical Interrupt Domain.

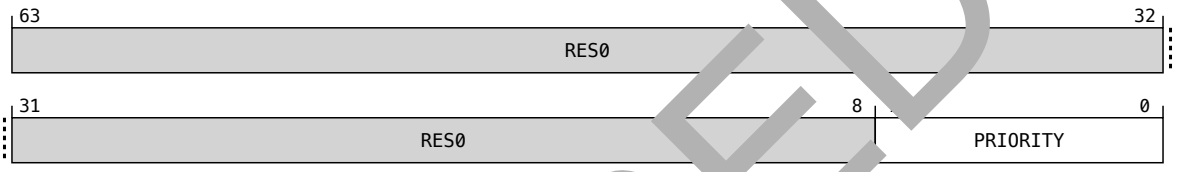
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_HAPR_EL1 are UNDEFINED.

**Attributes**

ICC_HAPR_EL1 is a 64-bit register.

#### Field descriptions




**Bits [63:8]**

Reserved, RES0.

**PRIORITY, bits [7:0]**

The running priority for the Current Physical Interrupt Domain.

If there are no active priorities in the CPU interface in the applicable Interrupt Domain, or all active priorities have undergone a priority drop, the value returned is the Idle priority.

#### Accessing ICC_HAPR_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_HAPR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b011 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_HAPR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_HAPR_EL1;
    else
        X[t, 64] = ICC_HAPR_EL1;
elsif PSTATE.EL == EL2 then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    X[t, 64] = ICC_HAPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_HAPR_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.7 ICC_HPPIR_EL1, Interrupt Controller Physical Highest Priority Pending Interrupt Register (EL1)

The ICC_HPPIR_EL1 characteristics are:

**Purpose**

Reports the HPPI for the Physical Interrupt Domain associated with the Security state selected by SCR_EL3.{NS, NSE}.

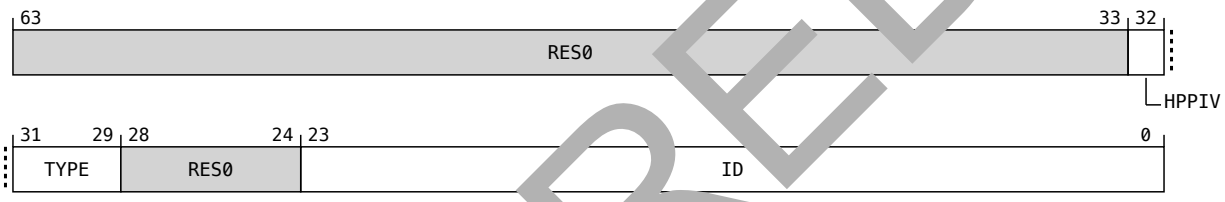
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_HPPIR_EL1 are UNDEFINED.

**Attributes**

ICC_HPPIR_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HPPIV, bit [32]**

HPPI valid.

There is an HPPI with Sufficient priority for the Interrupt Domain.

| HPPIV | Meaning |
|---|---|
| 0b0 | Invalid: There is no HPPI with Sufficient priority for the Interrupt Domain. |
| 0b1 | VALID: There is an HPPI with Sufficient priority for the Interrupt Domain. |

If ICC_HPPIR_EL1.HPPIV is 1, ID and TYPE together form the INTID of the HPPI for the Interrupt Domain.

**TYPE, bits [31:29]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICC_HPPIR_EL1.HPPIV is 0, TYPE is RES0.
- If ICC_HPPIR_EL1.HPPIV is 1, TYPE specifies the Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| TYPE | Meaning |
|---|---|
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICC_HPPIR_EL1.HPPIV is 0, ID is RES0.
- If ICC_HPPIR_EL1.HPPIV is 1, ID specifies the interrupt ID.

## Accessing ICC_HPPIR_EL1

This register is Read-only.

When accessed at EL3, if SCR_EL3.{NS, NSE} select a reserved value, the behavior is CONSTRAINED UNPREDICTABLE with the permitted options:

- SCR_EL3.{NS, NSE} is treated as having an UNKNOWN value.
- The access is treated as a NOP.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_HPPIR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b011 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_HPPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_HPPIR_EL1;
    else
        X[t, 64] = ICC_HPPIR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_HPPIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_HPPIR_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.8 ICC_HPPIR_EL3, Interrupt Controller Physical Highest Priority Pending Interrupt Register (EL3)

The ICC_HPPIR_EL3 characteristics are:

**Purpose**

Reports the HPPI for the EL3 Interrupt Domain.

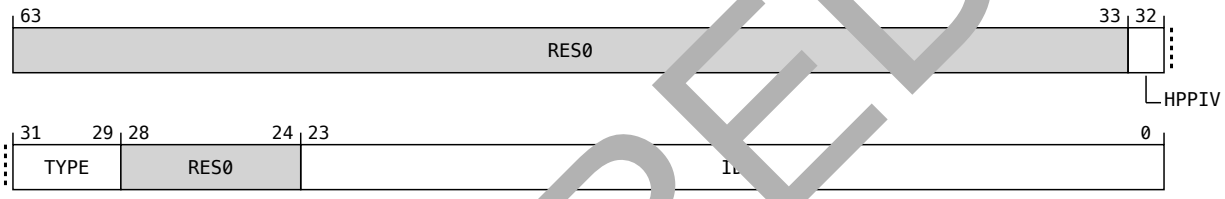
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_HPPIR_EL3 are UNDEFINED.

**Attributes**

ICC_HPPIR_EL3 is a 64-bit register.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HPPIV, bit [32]**

HPPI valid.

There is an HPPI with Sufficient priority for the Interrupt Domain.

| HPPIV | Meaning |
|---|---|
| 0b0 | Invalid: There is no HPPI with Sufficient priority for the Interrupt Domain. |
| 0b1 | VALID: There is an HPPI with Sufficient priority for the Interrupt Domain. |

If ICC_HPPIR_EL3.HPPIV is 1, ID and TYPE together form the INTID of the HPPI for the Interrupt Domain.

**TYPE, bits [31:29]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICC_HPPIR_EL3.HPPIV is 0, TYPE is RES0.
- If ICC_HPPIR_EL3.HPPIV is 1, TYPE specifies the Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICC_HPPIR_EL3.HPPIV is 0, ID is RES0.
- If ICC_HPPIR_EL3.HPPIV is 1, ID specifies the interrupt ID.

## Accessing ICC_HPPIR_EL3

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_HPPIR_EL3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1001 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_HPPIR_EL3;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.9 ICC_IAFFIDR_EL1, Interrupt Controller PE Interrupt Affinity ID Register

The ICC_IAFFIDR_EL1 characteristics are:

**Purpose**

Reports the PE interrupt Affinity ID for the PE.

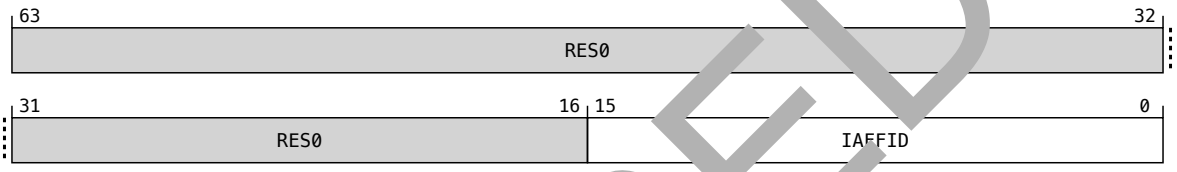
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_IAFFIDR_EL1 are UNDEFINED.

**Attributes**

ICC_IAFFIDR_EL1 is a 64-bit register.

#### Field descriptions




**Bits [63:16]**

Reserved, RES0.

**IAFFID, bits [15:0]**

PE interrupt Affinity ID.

#### Accessing ICC_IAFFIDR_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_IAFFIDR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b101 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_IAFFIDR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICC_IAFFIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_IAFFIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_IAFFIDR_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.10 ICC_ICSR_EL1, Interrupt Controller Interrupt Configuration and State Register

The ICC_ICSR_EL1 characteristics are:

**Purpose**

Reports the configuration and state of the INTID specified to a previous GIC request interrupt configuration instruction.

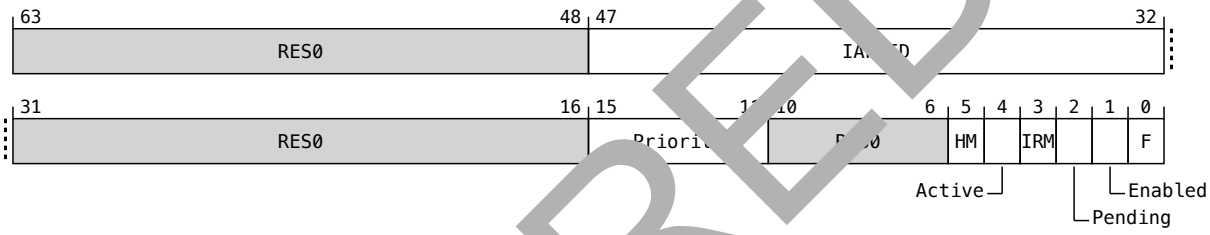
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_ICSR_EL1 are UNDEFINED.

**Attributes**

ICC_ICSR_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:48]**

Reserved, RES0.

**IAFFID, bits [47:32]**

The interrupt Affinity value.

The IRS may support fewer than 16 bits of IAFFID. Upper bits not implemented by the IRS are returned as zero.

When IRM is 1, this field is IMPLEMENTATION SPECIFIC.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [31:16]**

Reserved, RES0.

**Priority, bits [15:11]**

The priority of the interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [10:6]**

Reserved, RES0.

**HM, bit [5]**

The interrupt's Handling mode.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| HM | Meaning |
|---|---|
| 0b0 | Edge-triggered |
| 0b1 | Level-sensitive |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Active, bit [4]**

The interrupt's Active state.

| Active | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**IRM, bit [3]**

Interrupt Routing mode.

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted. |
| 0b1 | The interrupt Routing mode is 1ofN. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Pending, bit [2]**

The interrupt's Pending state.

| Pending | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Enabled, bit [1]**

The interrupt's individual enable.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| Enabled | Meaning |
|---|---|
| 0b0 | Disabled |
| 0b1 | Enabled |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**F, bit [0]**

Indicates whether the IRS returned valid data.

| F | Meaning |
|---|---|
| 0b0 | Request completed successfully. |
| 0b1 | Request did not complete successfully. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICC_ICSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_ICSR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b100 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_ICSR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICC_ICSR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_ICSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_ICSR_EL1;
```

**MSR ICC_ICSR_EL1, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b100 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_ICSR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICC_ICSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_ICSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_ICSR_EL1 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.11 ICC_IDR0_EL1, Interrupt Controller ID Register 0

The ICC_IDR0_EL1 characteristics are:

**Purpose**

Contains information about features implemented by the GICv5 CPU interface.

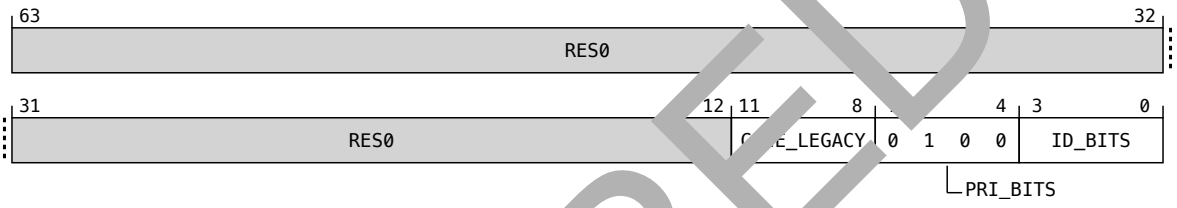
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_IDR0_EL1 are UNDEFINED.

**Attributes**

ICC_IDR0_EL1 is a 64-bit register.

#### Field descriptions




**Bits [63:12]**

Reserved, RES0.

**GCIE_LEGACY, bits [11:8]**

Indicates support for legacy GICv3.3 virtual CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| GCIE_LEGACY | Meaning |
|---|---|
| 0b0000 | Legacy GICv3.3 virtual CPU interface is not implemented |
| 0b0001 | Legacy GICv3.3 virtual CPU interface is implemented |

FEAT_GCIE_LEGACY extension implements the functionality identified by the value 0b0001.

Access to this field is RO.

**PRI_BITS, bits [7:4]**

The number of priority bits implemented, minus one.

The number of supported priority bits is 5.

Reads as 0b0100

Other values are reserved.

Access to this field is RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**ID_BITS, bits [3:0]**

Identifier bits.

The number of interrupt identifier bits supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| ID_BITS | Meaning |
|---|---|
| 0b0000 | 16 bits |
| 0b0001 | 24 bits |

Access to this field is RO.

## Accessing ICC_IDR0_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_IDR0_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b010 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_HCR_EL2.TDIR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_VGRTR_EL2.ICC_IDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICC_IDR0_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_IDR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_IDR0_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.12 ICC_PCR_EL1, Interrupt Controller Physical Interrupt Priority Control Register (EL1)

The ICC_PCR_EL1 characteristics are:

**Purpose**

Controls the Physical priority mask for the Non-secure, Realm, and Secure Interrupt Domains.

**Configuration**

There are separate banked copies of this register for Non-secure, Realm and Secure state.

This register is banked between ICC_PCR_EL1 and ICC_PCR_EL1_NS and ICC_PCR_EL1_RL and ICC_PCR_EL1_S.

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PCR_EL1 are UNDEFINED.

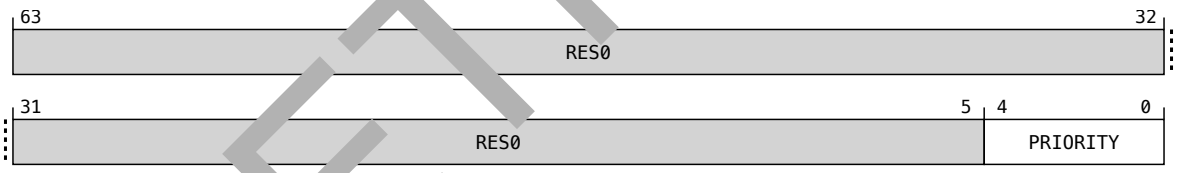
**Attributes**

ICC_PCR_EL1 is a 64-bit register.

This register has the following instances:

- ICC_PCR_EL1, when EL3 is not implemented.
- ICC_PCR_EL1_NS, when EL3 is implemented.
- ICC_PCR_EL1_RL, when FEAT_RME is implemented.
- ICC_PCR_EL1_S, when (HaveEL(EL3) && (!IsFeatureImplemented(FEAT_RME))) || (IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SEL2)).

#### Field descriptions




**Bits [63:5]**

Reserved, RES0.

**PRIORITY, bits [4:0]**

The priority mask for the Interrupt Domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

#### Accessing ICC_PCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PCR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b010 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PCR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PCR_EL1;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_PCR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_PCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_PCR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_PCR_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_PCR_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_PCR_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_PCR_EL1_NS;
    else
        UNDEFINED;
```

## MSR ICC_PCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b010 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PCR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PCR_EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            ICC_PCR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_PCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_PCR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_PCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_PCR_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        ICC_PCR_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        ICC_PCR_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.2.13 ICC_PCR_EL3, Interrupt Controller Interrupt Priority Control Register (EL3)

The ICC_PCR_EL3 characteristics are:

**Purpose**

Controls the Physical priority mask for the physical CPU interface for the EL3 Interrupt Domain.

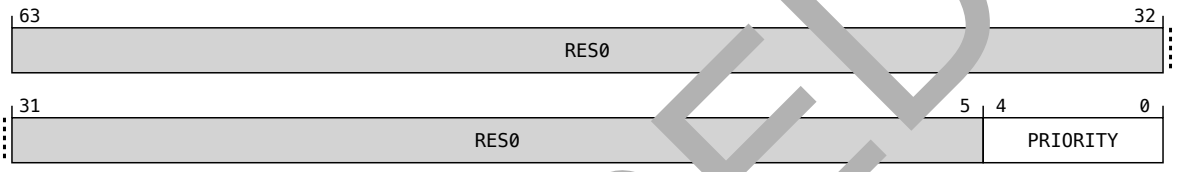
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_PCR_EL3 are UNDEFINED.

**Attributes**

ICC_PCR_EL3 is a 64-bit register.

## Field descriptions




**Bits [63:5]**

Reserved, RES0.

**PRIORITY, bits [4:0]**

The priority mask for the EL3 Interrupt Domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICC_PCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PCR_EL3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1000 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PCR_EL3;
```

**MSR ICC_PCR_EL3, <Xt>**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1000 | 0b001 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_PCR_EL3 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.2.14 ID_AA64PFR2_EL1, AArch64 Processor Feature Register 2

The ID_AA64PFR2_EL1 characteristics are:

**Purpose**

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

**Configuration**

---

**Note**

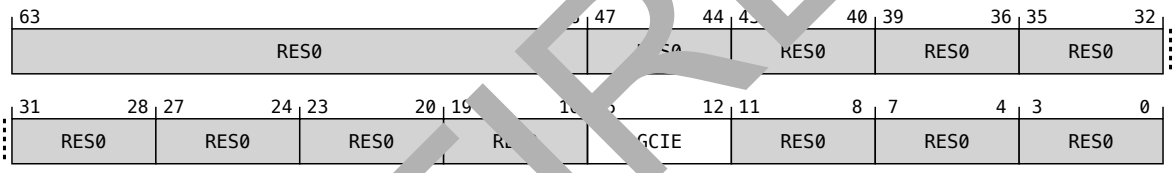
Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

---

**Attributes**

ID_AA64PFR2_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:48]**

Reserved, RES0.

**Bits [47:44]**

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**Bits [39:36]**

Reserved, RES0.

**Bits [35:32]**

Reserved, RES0.

**Bits [31:28]**

Reserved, RES0.

**Bits [27:24]**

Reserved, RES0.

**Bits [23:20]**

Reserved, RES0.

**Bits [19:16]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**GCIE, bits [15:12]**

Support for the GICv5 CPU interface extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| GCIE | Meaning |
|---|---|
| 0b0000 | FEAT_GCIE is not supported. |
| 0b0001 | FEAT_GCIE is supported. |

Access to this field is RO.

**Bits [11:8]**

Reserved, RES0.

**Bits [7:4]**

Reserved, RES0.

**Bits [3:0]**

Reserved, RES0.

## Accessing ID_AA64PFR2_EL1

Accesses to this register use the following encoding in the System register encoding space:

**MRS <Xt>, ID_AA64PFR2_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !IsZero(ID_AA64PFR2_EL1) || boolean IMPLEMENTATION_DEFINED "ID_AA64PFR2_EL1
      ↪trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR2_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.3 Virtual CPU interface registers

$I_{TRMKL}$ Each ICC_ System register that is accessible at EL1 and higher and whose state is specific to the Virtual Interrupt Domain, has a corresponding virtual ICV_ register. The ICV_ registers are accessed using the same System register encodings as their ICC_ counterparts.

$I_{TBPCJ}$ ICC_IDR0 does not have an ICV_ equivalent register.

$I_{LBWLV}$ When Legacy operation is disabled, the field layouts and interpretations of the ICV_ registers are identical to the ICC_ registers. This enables binary compatibility for software between the physical and virtual CPU interfaces.

$I_{MCZFN}$ ICC_ICSR_EL1 does not have an ICV_ equivalent register.

$I_{NMXVC}$ ICC_IAFFIDR_EL1 does not have an ICV_ equivalent register. When virtualization is being used, Arm expects EL2 software to trap EL1 accesses to ICC_IAFFIDR_EL1.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.3.1 ICV_APR_EL1, Interrupt Controller Virtual Active Priorities Register

The ICV_APR_EL1 characteristics are:

**Purpose**

Records active priorities for the Virtual CPU interface.

**Configuration**

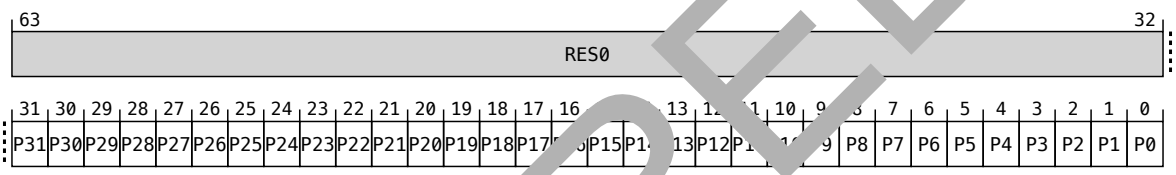
AArch64 System register ICV_APR_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_APR_EL2[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_APR_EL1 are UNDEFINED.

**Attributes**

ICV_APR_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**P<x> , bits [x], for x = 31 to**

Provides access to the active pri ities.

| P<x> | Meaning |
|---|---|
| 0b0 | Priority not active |
| 0b1 | Priority active |

Fie in this register are indexed using the 5-bit priority as an unsigned integer, P[Priority].

The rese ehavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICV_APR_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_APR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b000 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_APR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_APR_EL1;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_APR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_APR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_APR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_APR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_APR_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_APR_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_APR_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_APR_EL1_NS;
    else
        UNDEFINED;
```

## MSR ICC_APR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b000 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_APR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_APR_EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            ICC_APR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_APR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_APR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_APR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_APR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_APR_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.[illegible] == [illegible] && SCR_EL3.NS == '1' then
        ICC_APR_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == [illegible]then
        ICC_APR_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.3.2 ICV_CR0_EL1, Interrupt Controller EL1 Virtual Control Register

The ICV_CR0_EL1 characteristics are:

**Purpose**

Controls behavior of the CPU interface in the Virtual Interrupt Domain.

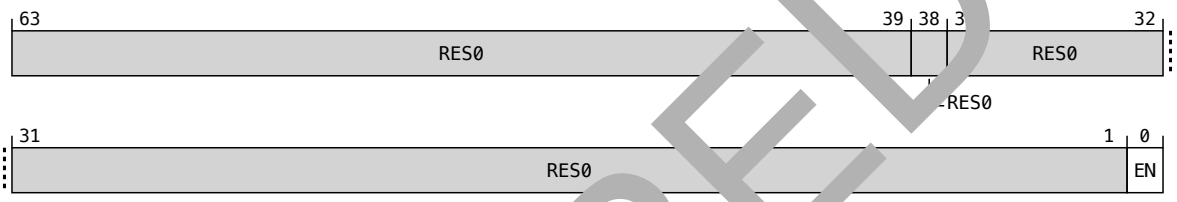
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_CR0_EL1 are UNDEFINED.

**Attributes**

ICV_CR0_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:39]**

Reserved, RES0.

**Bit [38]**

Reserved, RES0.

**Bits [37:32]**

Reserved, RES0.

**Bits [31:1]**

Reserved, RES0.

**EN, bit [0]**

Enable interrupts for the Interrupt Domain.

When this field is 0, there is no HPPI of Sufficient priority for the Interrupt Domain.

| EN | Meaning |
|---|---|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Accessing ICV_CR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_CR0_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b001 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_CR0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_CR0_EL1;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CR0_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_CR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CR0_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_CR0_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_CR0_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_CR0_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_CR0_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        X[t, 64] = ICC_CR0_EL1_NS;
    else
        UNDEFINED;
```

**MSR ICC_CR0_EL1, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b001 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_CR0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_CR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CR0_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_CR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CR0_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_CR0_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_CR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_CR0_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        ICC_CR0_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        ICC_CR0_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.3.3 ICV_HAPR_EL1, Interrupt Controller Virtual Highest Active Priority Register

The ICV_HAPR_EL1 characteristics are:

**Purpose**

Reports the running priority of the Virtual Interrupt Domain.

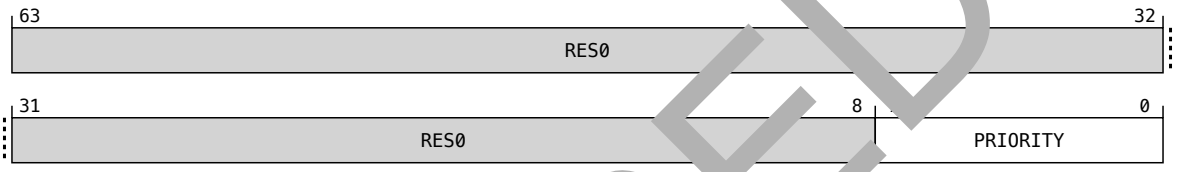
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HAPR_EL1 are UNDEFINED.

**Attributes**

ICV_HAPR_EL1 is a 64-bit register.

#### Field descriptions




**Bits [63:8]**

Reserved, RES0.

**PRIORITY, bits [7:0]**

The running priority for the Virtual Interrupt Domain.

If there are no active priorities in the CPU interface in the applicable Interrupt Domain, or all active priorities have undergone a priority drop, the value returned is the Idle priority.

#### Accessing ICV_HAPR_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_HAPR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b011 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_HAPR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_HAPR_EL1;
    else
        X[t, 64] = ICC_HAPR_EL1;
elsif PSTATE.EL == EL2 then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    X[t, 64] = ICC_HAPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_HAPR_EL1;
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.3.4 ICV_HPPIR_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register

The ICV_HPPIR_EL1 characteristics are:

**Purpose**

Reports the HPPI for the Virtual Interrupt Domain.

**Configuration**

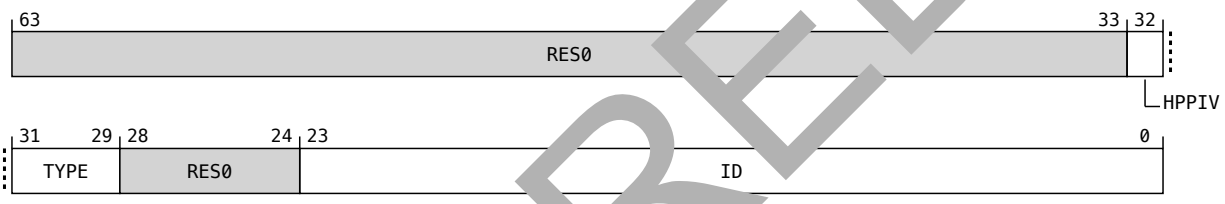
AArch64 System register ICV_HPPIR_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_HPPIR_EL2[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR_EL1 are UNDEFINED.

**Attributes**

ICV_HPPIR_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HPPIV, bit [32]**

HPPI valid.

There is an HPPI with sufficient priority for the Interrupt Domain.

| HPPIV | Meaning |
|---|---|
| 0b0 | Invalid: There is no HPPI with Sufficient priority for the Interrupt Domain. |
| 0b1 | VALID: There is an HPPI with Sufficient priority for the Interrupt Domain. |

If ICV_HPPIR_EL1.HPPIV is 1, ID and TYPE together form the INTID of the HPPI for the Interrupt Domain.

**TYPE, bits [31:29]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICV_HPPIR_EL1.HPPIV is 0, TYPE is RES0.
- If ICV_HPPIR_EL1.HPPIV is 1, TYPE specifies the Type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| TYPE | Meaning |
|---|---|
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICV_HPPIR_EL1.HPPIV is 0, ID is RES0.
- If ICV_HPPIR_EL1.HPPIV is 1, ID specifies the interrup[illegible]D.

## Accessing ICV_HPPIR_EL1

This register is Read-only.

Accesses to this register use the following encodings in the [illegible]register encoding space:

**MRS <Xt>, ICC_HPPIR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b[illegible]0 | 0b1010 | 0b011 |

```
if !IsFeatureImplemented[illegible]EAT_GC[illegible]) then
    UNDEFINED;
elsif PSTATE.EL == [illegible] then
    UNDEFINED;
elsif PSTATE.EL[illegible] = EL1 th[illegible]
    if EL2Enable[illegible]() && [illegible]R_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VC[illegible]2.V3 == [illegible] then
        U[illegible]NED;
    el[illegible] EL2E[illegible]abled()[illegible] && [illegible]_HFGRTR_EL2.ICC_HPPIR_EL1 == '0' then
        AArch6[illegible].SystemAc[illegible]sTrap(EL2, 0x18);
    [illegible]sif EL[illegible]abled() && HCR_EL2.IMO == '1' then
        X[illegible]64] = [illegible]_HPPIR_EL1;
    els[illegible]
        X[illegible] 64] = ICC_HPPIR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_HPPIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_HPPIR_EL1;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.3.5 ICV_PCR_EL1, Interrupt Controller Virtual Interrupt Priority Control Register

The ICV_PCR_EL1 characteristics are:

**Purpose**

Controls the Virtual priority mask for the Virtual Interrupt Domain.

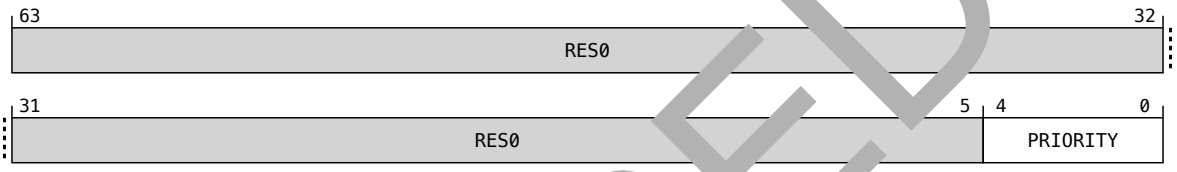
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PCR_EL1 are UNDEFINED.

**Attributes**

ICV_PCR_EL1 is a 64-bit register.

#### Field descriptions




**Bits [63:5]**

Reserved, RES0.

**PRIORITY, bits [4:0]**

The priority mask for the Interrupt Domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

#### Accessing ICV_PCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICV_PCR_EL1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b010 |

```
if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PCR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PCR_EL1;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_PCR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            X[t, 64] = ICC_PCR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_PCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_PCR_EL1_S;
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_RL;
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            X[t, 64] = ICC_PCR_EL1_NS;
        else
            UNDEFINED;
    else
        X[t, 64] = ICC_PCR_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_PCR_EL1_S;
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.N  == '1' && SC  L3  S == '1' then
        X[t, 64] = ICC_PCR_EL1_RL;
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' th
        X[t, 64] = ICC_PCR_EL1_NS;
    else
        UNDEFINED;
```

## MSR ICC_PCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b001 | 0b1100 | 0b0000 | 0b010 |

```
if !IsFeatureImp  mented(F AT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL    EL0  hen
    UNDEFINED;
elsif PS    == E  then
    if  L2Enab  d() &&  P  L2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
     ↪ICH   CTLR_EL2.  == '1' then
    UNDE
    el   2Enabled() && ICH_HFGWTR_EL2.ICC_PCR_EL1 == '0' then
        ch64.SystemAccessTrap(EL2, 0x18);
    elsif   Enabled() && HCR_EL2.IMO == '1' then
        ICV_  _EL1 = X[t, 64];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_PCR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_PCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_PCR_EL1_S = X[t, 64];
        elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
            ICC_PCR_EL1_RL = X[t, 64];
        elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            ICC_PCR_EL1_NS = X[t, 64];
        else
            UNDEFINED;
    else
        ICC_PCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_PCR_EL1_S = X[t, 64];
    elsif IsFeatureImplemented(FEAT_RME) && SCR_EL3.NSE == '1' && SCR_EL3.NS == '1' then
        ICC_PCR_EL1_RL = X[t, 64];
    elsif SCR_EL3.NSE == '0' && SCR_EL3.NS == '1' then
        ICC_PCR_EL1_NS = X[t, 64];
    else
        UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.4 PPI registers

Configuration and state of PPIs.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.1 ICC_PPI_CACTIVER<n>_EL1, Interrupt Controller Physical PPI Clear Active Registers, n = 0 - 1

The ICC_PPI_CACTIVER<n>_EL1 characteristics are:

**Purpose**

Clear Active state for physical PPIs.

**Configuration**

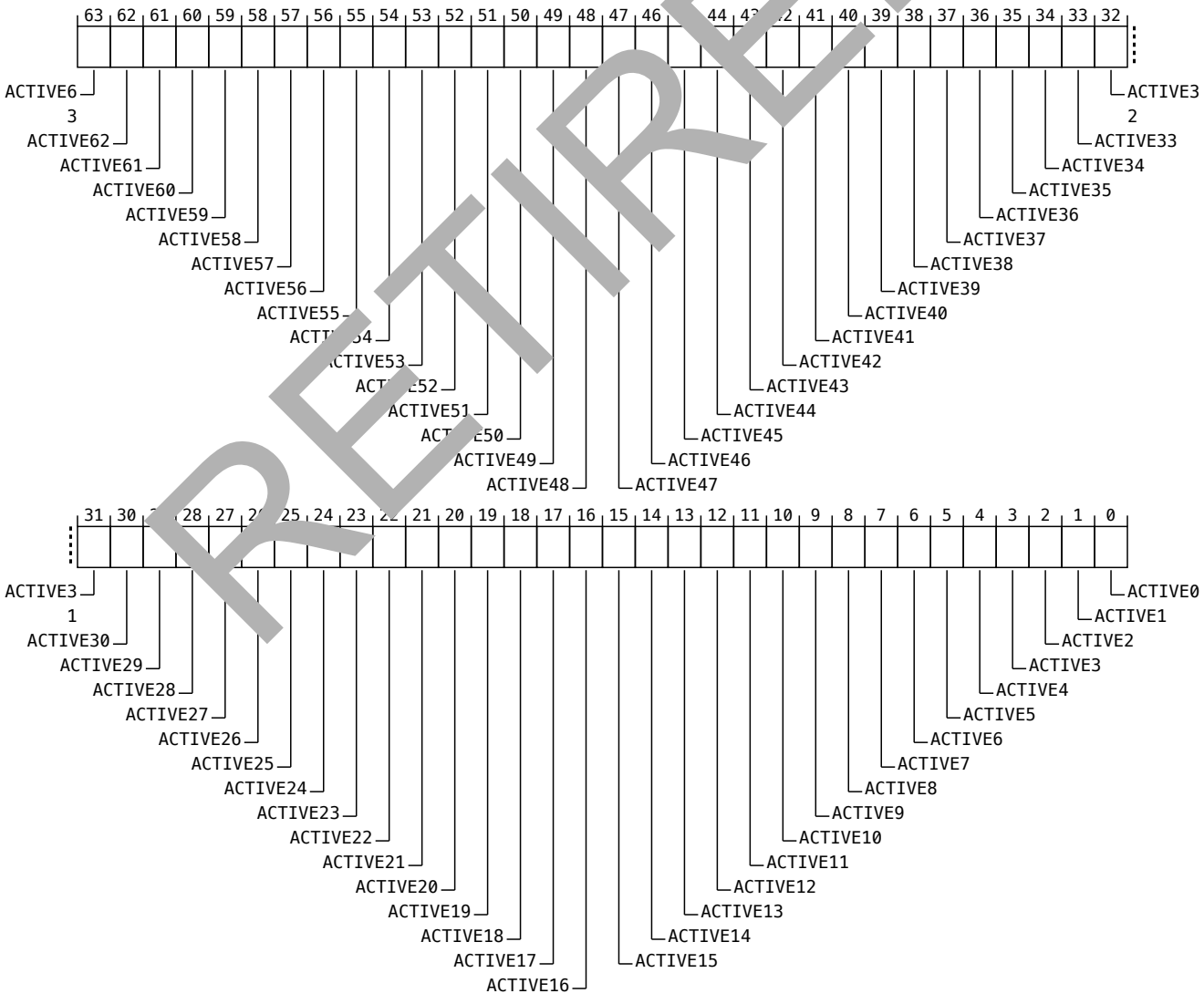
AArch64 System register ICC_PPI_CACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICC_PPI_SACTIVER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_CACTIVER<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_CACTIVER<n>_EL1 is a 64-bit register.

## Field descriptions




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**ACTIVE<x> , bits [x], for x = 63 to 0**

Configures whether PPIs are Active.

Reads return the Active state of the INTID.

Writing 1 clears the Active state of the INTID. Writing 0 has no effect.

| ACTIVE<x> | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !IsPPIAssignedToCurrentDomain((n * 64) + x)
  - PSTATE.EL IN {EL2, EL1}
- Otherwise, access to this field is WI.

## Accessing ICC_PPI_CACTIVER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_CACTIVER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b00:n[0] |

```
integer t = UInt(op2<0>);

if IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_CACTIVER_EL1[n];
    else
        X[t, 64] = ICC_PPI_CACTIVER_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_CACTIVER_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_CACTIVER_EL1[n];
```

**MSR ICC_PPI_CACTIVER<n>_EL1, <Xt> ; Where n = 0-1**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b00:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_CACTIVER_EL1[n] = X[t, 64];
    else
        ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.2 ICC_PPI_CPENDR<n>_EL1, Interrupt Controller Physical PPI Clear Pending State Registers, n = 0 - 1

The ICC_PPI_CPENDR<n>_EL1 characteristics are:

**Purpose**

Clear pending state for physical PPIs.

**Configuration**

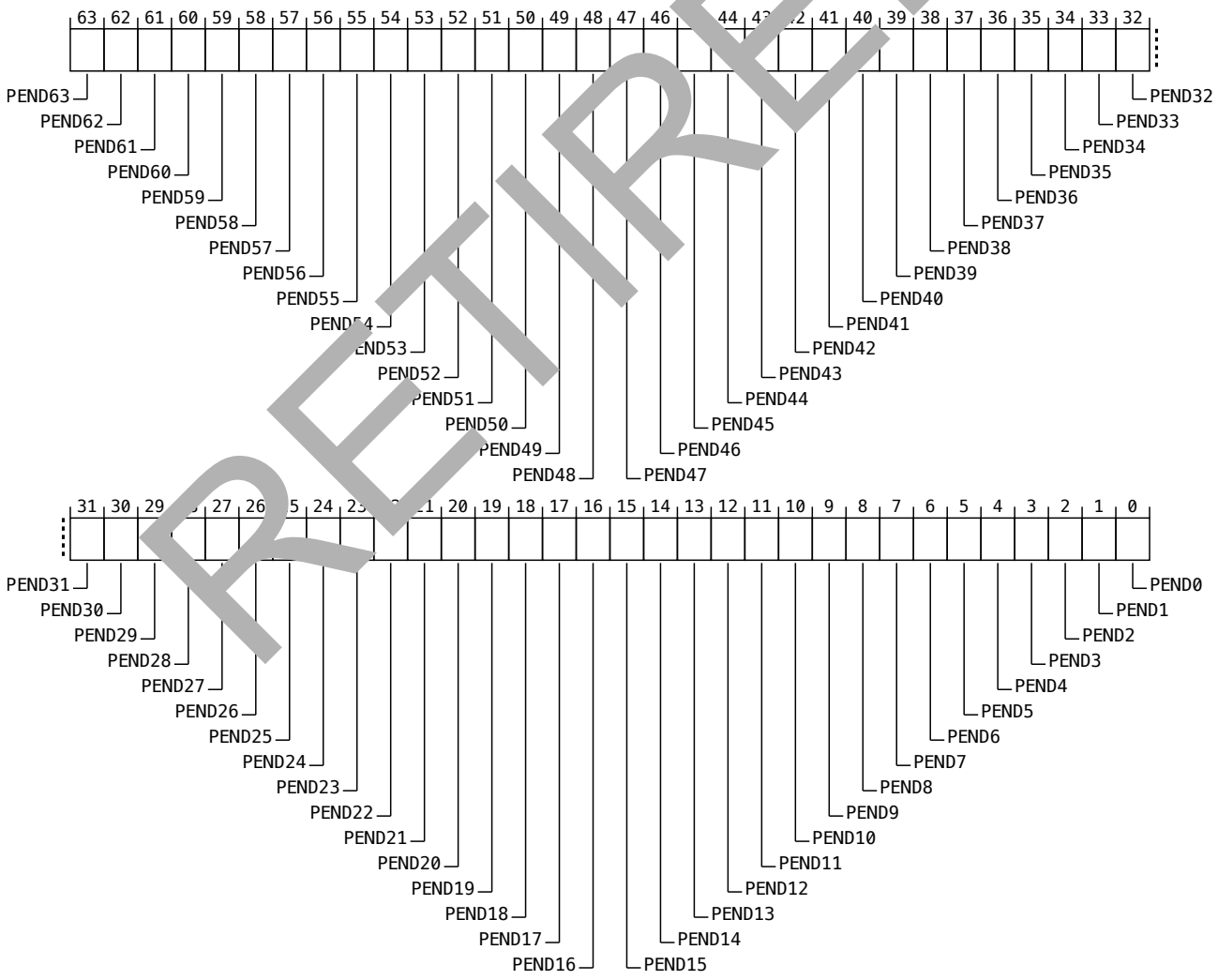
AArch64 System register ICC_PPI_CPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICC_PPI_SPENDR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_CPENDR<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_CPENDR<n>_EL1 is a 64-bit register.

## Field descriptions




**PEND<x> , bits [x], for x = 63 to 0**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Controls the Pending state of PPIs.

Reads return the current state of the INTIDs.

Writing 1 to a field clears the Pending state of the corresponding INTID. Writing 0 has no effect.

| PEND<x> | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !IsPPIAssignedToCurrentDomain((n * 64) + x)
  - PSTATE.EL IN {EL2, EL1}
- When ICC_PPI_HMR_EL1[n].HM<x> == 1, access to this field is RO.
- Otherwise, access to this field is W1C.

## Accessing ICC_PPI_CPENDR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_CPENDR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PENDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_CPENDR_EL1[n];
    else
        X[t, 64] = ICC_PPI_CPENDR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_CPENDR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_CPENDR_EL1[n];
```

**MSR ICC_PPI_CPENDR<n>_EL1, <Xt> ; Where n = 0-1**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_PENDRn_EL1 == '    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_CPENDR_EL1[n] = X[t, 64];
    else
        ICC_PPI_CPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_CPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_CPENDR_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.3 ICC_PPI_DOMAINR<n>_EL3, Interrupt Controller PPI Domain Registers, n = 0 - 3

The ICC_PPI_DOMAINR<n>_EL3 characteristics are:

**Purpose**

Controls which Interrupt Domain PPI sources are assigned to.

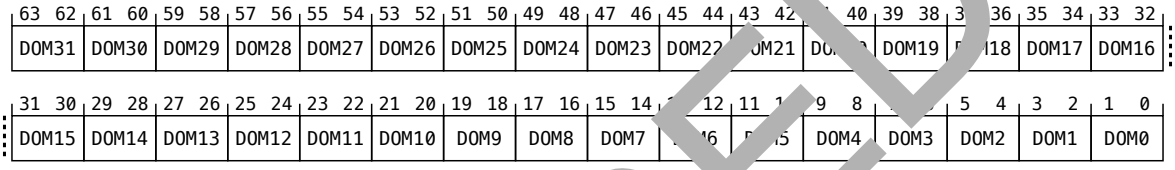
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_PPI_DOMAINR<n>_EL3 are UNDEFINED.

**Attributes**

ICC_PPI_DOMAINR<n>_EL3 is a 64-bit register.

## Field descriptions




**DOM<x> , bits [2x+1:2x], for x = 31 to 0**

Controls the Physical Interrupt Domain that a PPI is assigned to.

If a physical PPI is assigned to an unimplemented Interrupt Domain, all of the following are true:

- The physical PPI is not selected as the candidate HPPI in any Interrupt Domain.
- Access to the state and configuration of the PPI using an ICC_PPI_ System register is only permitted at EL3.
- Attempts to deactivate the PPI by executing a GIC xDDI System instruction are treated as NOPs.

| DOM<x> | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 32) + x), access to this field is RES0.
- Otherwise, access to this field is RW.

## Accessing ICC_PPI_DOMAINR<n>_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_PPI_DOMAINR<n>_EL3 ; Where n = 0-3
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```
integer n = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_DOMAINR_EL3[n];
```

**MSR ICC_PPI_DOMAINR<n>_EL3, <Xt> ; Where n = 0–1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b110 | 0b1100 | 0b10[illegible] | 0b1:n[1:0] |

```
integer n = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_PPI_DOMAINR_EL3[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.4 ICC_PPI_ENABLER<n>_EL1, Interrupt Controller Physical PPI Enable Registers, n = 0 - 1

The ICC_PPI_ENABLER<n>_EL1 characteristics are:

**Purpose**

Access to Enable state for physical PPIs.

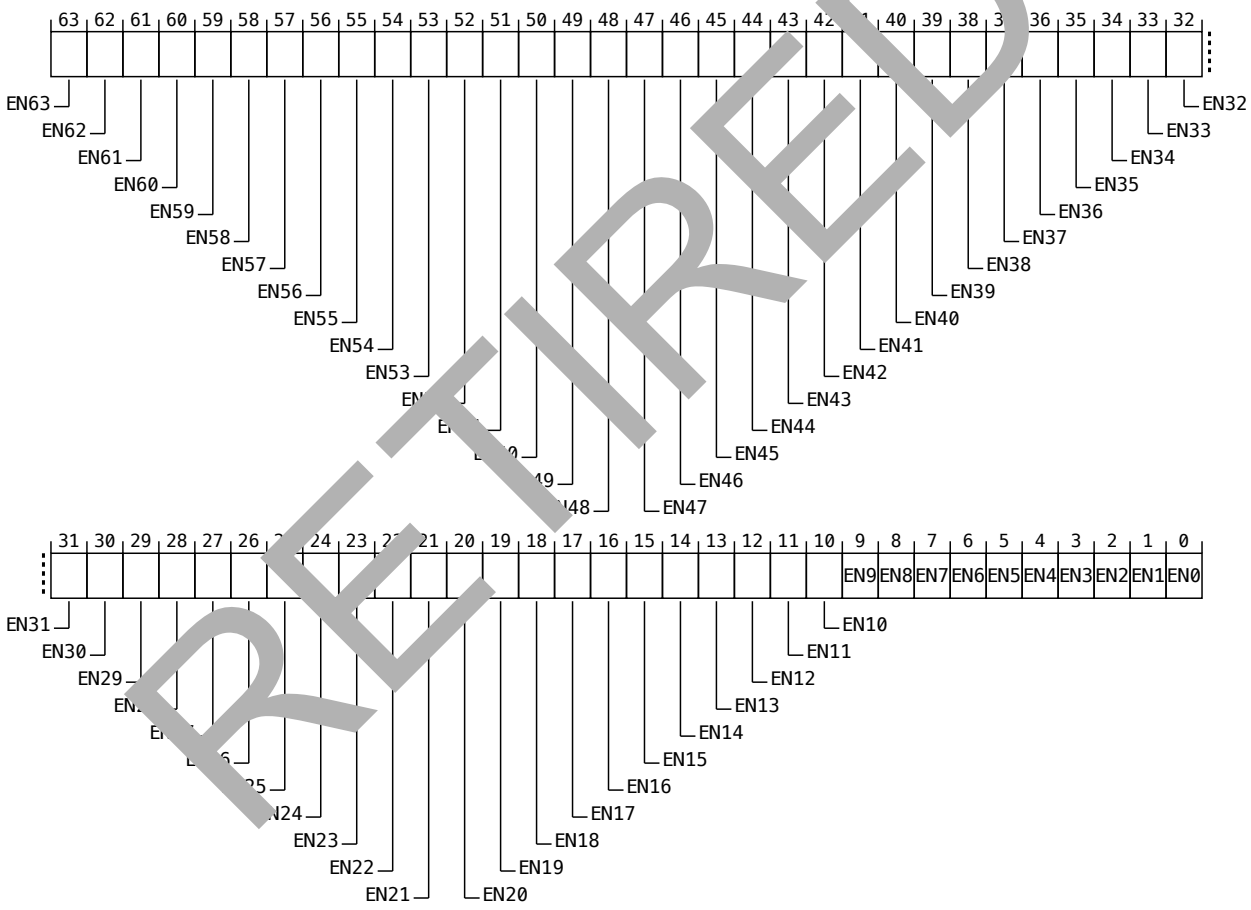
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_ENABLER<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_ENABLER<n>_EL1 is a 64-bit register.

## Field descriptions




**EN<x> , bits [x], for x = 63 to 0**

Configures whether PPIs are enabled.

Reads return the current state of the INTID.

| EN<x> | Meaning |
|---|---|
| 0b0 | Disabled |
| 0b1 | Enabled |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !IsPPIAssignedToCurrentDomain((n * 64) + x)
  - PSTATE.EL IN {EL2, EL1}
- Otherwise, access to this field is RW.

### Accessing ICC_PPI_ENABLER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_ENABLER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ENABLERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_ENABLER_EL1[n];
    else
        X[t, 64] = ICC_PPI_ENABLER_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_ENABLER_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_ENABLER_EL1[n];
```

**MSR ICC_PPI_ENABLER<n>_EL1, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ENABLERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_ENABLER_EL1[n] = X[t, 64];
    else
        ICC_PPI_ENABLER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_ENABLER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_ENABLER_EL1[n] = X[t, 64];
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.4.5 ICC_PPI_HMR<n>_EL1, Interrupt Controller Physical PPI Handling mode Registers, n = 0 - 1

The ICC_PPI_HMR<n>_EL1 characteristics are:

**Purpose**

Report whether physical PPIs are Edge or Level.

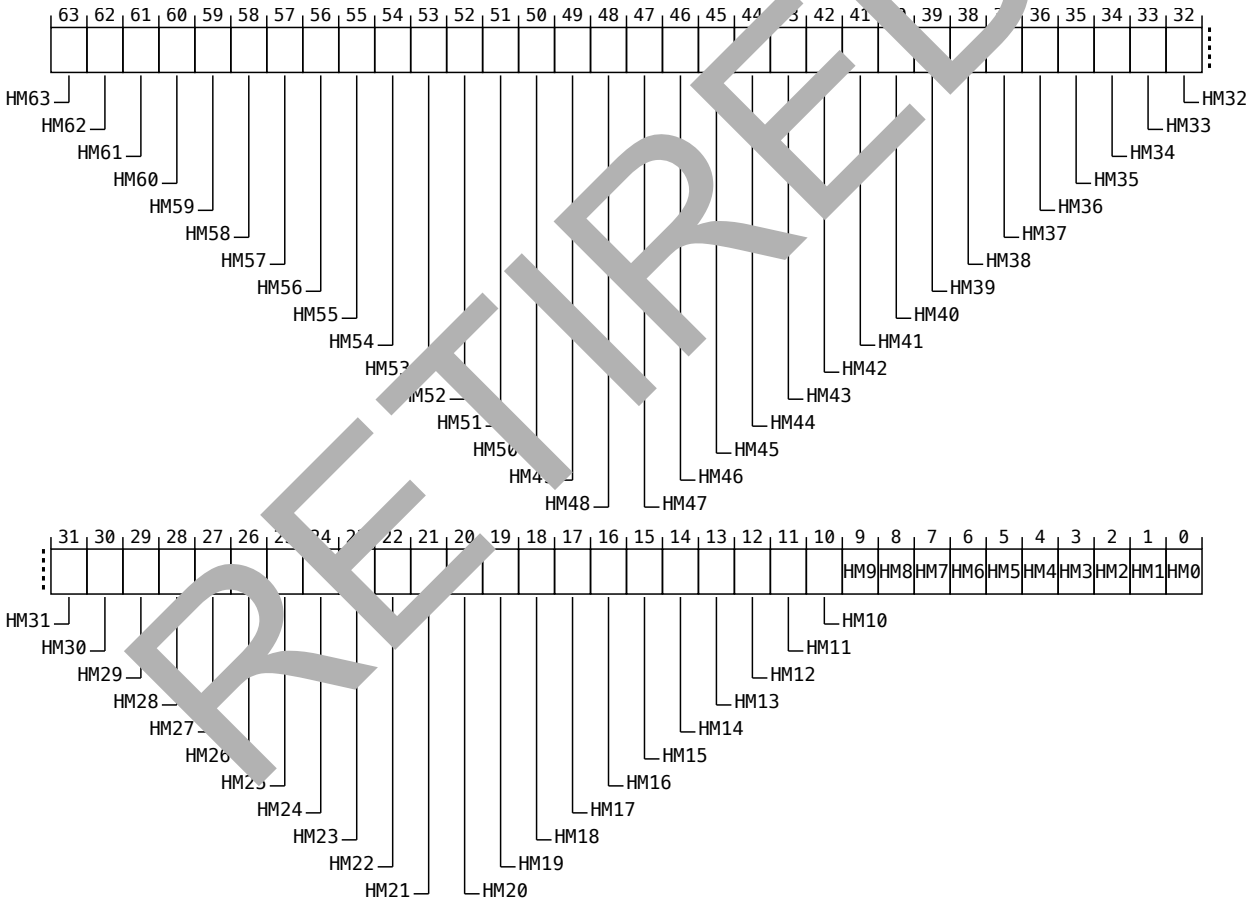
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_HMR<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_HMR<n>_EL1 is a 64-bit register.

## Field descriptions




**HM<x> , bits [x], for x = 63 to 0**

The PPI Handling mode.

| HM<x> | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | Level |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RO.

### Accessing ICC_PPI_HMR<n>_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_HMR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 00:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_HMRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_HMR_EL1[n];
    else
        X[t, 64] = ICC_PPI_HMR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_HMR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_HMR_EL1[n];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.6 ICC_PPI_PRIORITYR<n>_EL1, Interrupt Controller Physical PPI Priority Registers, n = 0 - 15

The ICC_PPI_PRIORITYR<n>_EL1 characteristics are:

**Purpose**

Configures the priority of physical PPIs.

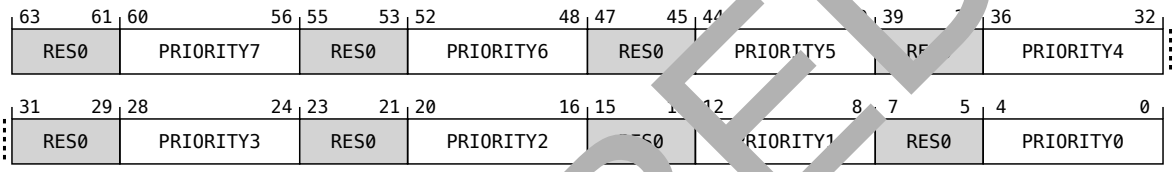
**Configuration**

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_PRIORITYR<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_PRIORITYR<n>_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:61, 55:53, 47:45, 39:37, 31:29, 23:21, 15:13, 7:5]**

Reserved, RES0.

**PRIORITY<x> , bits [60:56, 52:48, 44:40, 36:32, 28:24, 20:16, 12:8, 4:0], for x = 7 to 0**

Configures the priority for the corresponding PPI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 8) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !CanPPIAssignInterruptToCurrentDomain((n * 8) + x)
  - PSTATE.EL IN {EL2, EL1}
- Otherwise, access to this field is RW.

## Accessing ICC_PPI_PRIORITYR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_PRIORITYR<n>_EL1 ; Where n = 0–15**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b111:n[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PRIORITYRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_PRIORITYR_EL1[n];
    else
        X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
```

## MSR ICC_PPI_PRIORITYR<n>_EL1, <Xt> ; Where = 0-15

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b11:n[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.[illegible] == '[illegible]' && [illegible]atureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == [illegible]1' then
        UNDEFINED;
    elsif EL2Enabled() && [illegible]CH_[illegible]WTR_EL2.ICC_[illegible]I_PRIORITYRn_EL1 == '0' then
        AArch64.SystemAccessTrap[illegible]2, 0x18);
    elsif EL2Enabled([illegible] && HCR_EL2.[illegible] == '1' then
        ICV_PPI_PR[illegible]ITYR_EL1[n] = X[illegible]64];
    else
        ICC_PP[illegible]RIORITY[illegible]EL1[n] = X[t, 64];
elsif PSTATE.EL[illegible]EL2[illegible]hen
    ICC_PPI_PRIOR[illegible]EL1[n] = [illegible]t, 64];
elsif PS[illegible] == E[illegible]then
    IC[illegible]PPI_PR[illegible]ORITYR_[illegible] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.7 ICC_PPI_SACTIVER<n>_EL1, Interrupt Controller Physical PPI Set Active Registers, n = 0 - 1

The ICC_PPI_SACTIVER<n>_EL1 characteristics are:

**Purpose**

Set Active state for physical PPIs.

**Configuration**

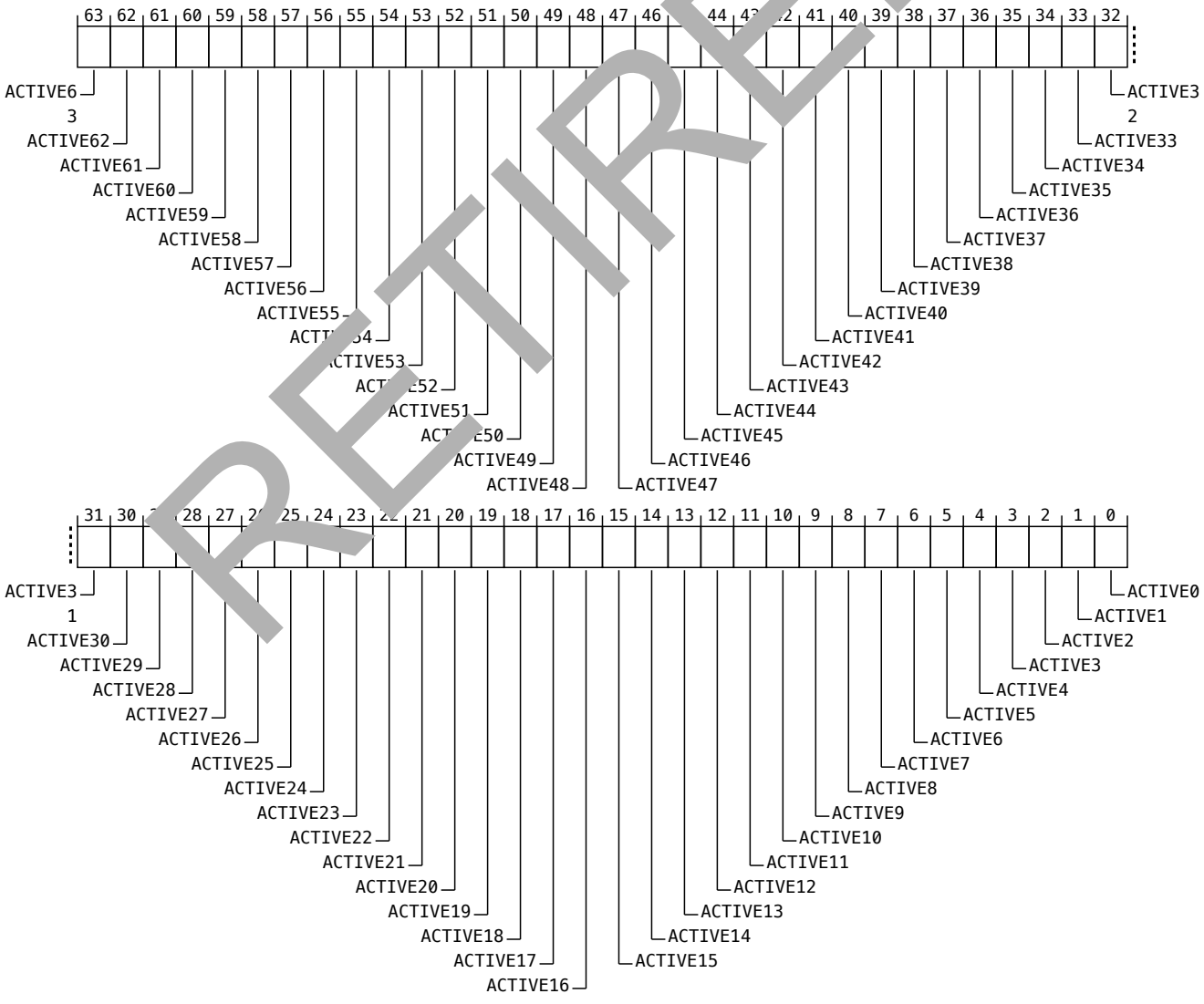
AArch64 System register ICC_PPI_SACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICC_PPI_CACTIVER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_SACTIVER<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_SACTIVER<n>_EL1 is a 64-bit register.

## Field descriptions




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**ACTIVE<x> , bits [x], for x = 63 to 0**

Configures whether PPIs are Active.

Reads return the Active state of the INTID.

Writing 1 sets the Active state of the INTID. Writing 0 has no effect.

| ACTIVE<x> | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !IsPPIAssignedToCurrentDomain((n * 64) + x)
  - PSTATE.EL IN {EL2, EL1}
- Otherwise, access to this field is W1.

## Accessing ICC_PPI_SACTIVER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_SACTIVER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b01:n[0] |

```
integer t = UInt(op2<0>);

if IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_SACTIVER_EL1[n];
    else
        X[t, 64] = ICC_PPI_SACTIVER_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_SACTIVER_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_SACTIVER_EL1[n];
```

**MSR ICC_PPI_SACTIVER<n>_EL1, <Xt> ; Where n = 0-1**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b01:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_SACTIVER_EL1[n] = X[t, 64];
    else
        ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.4.8 ICC_PPI_SPENDR<n>_EL1, Interrupt Controller Physical PPI Set Pending State Registers, n = 0 - 1

The ICC_PPI_SPENDR<n>_EL1 characteristics are:

**Purpose**

Set pending state for Physical PPIs.

**Configuration**

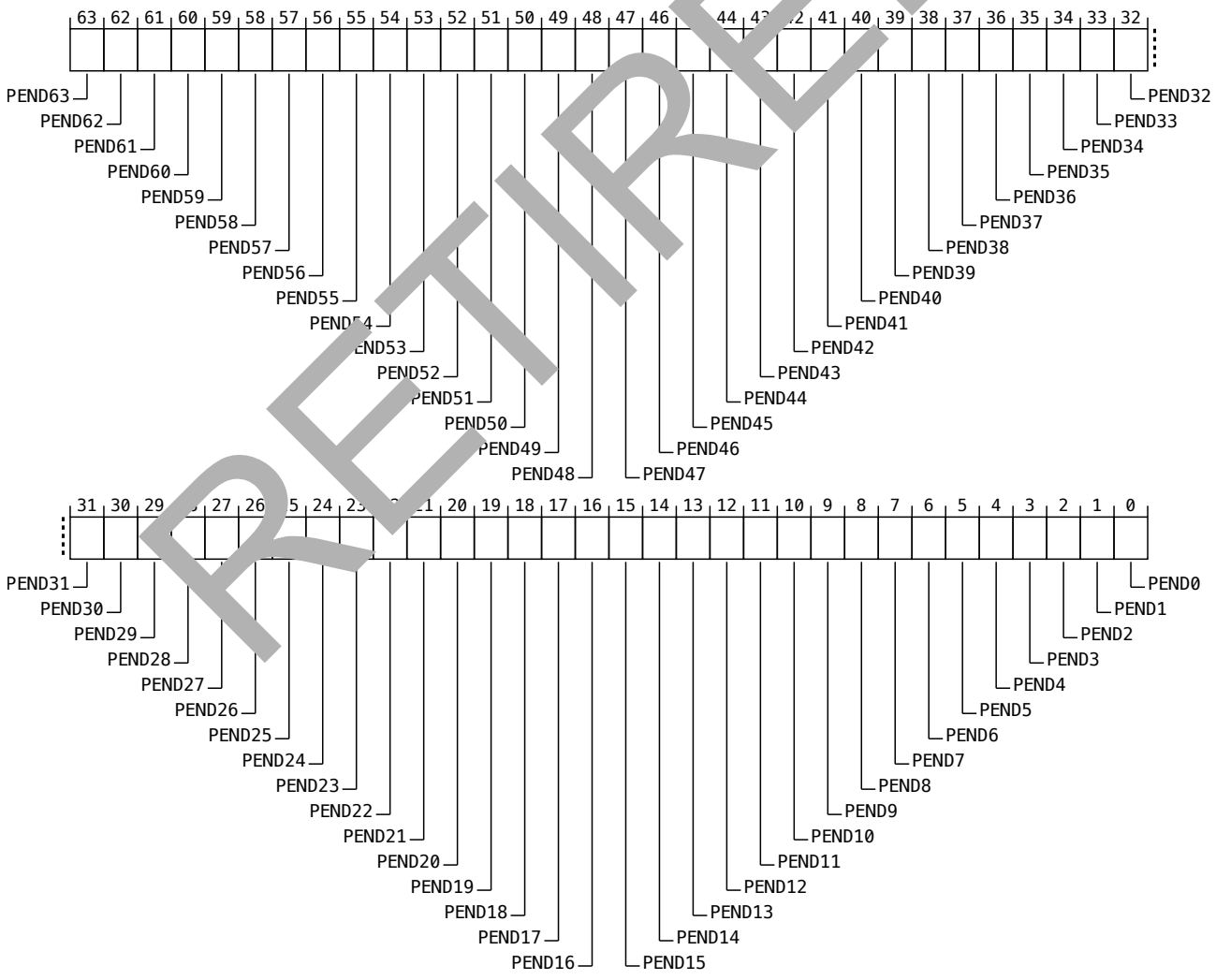
AArch64 System register ICC_PPI_SPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICC_PPI_CPENDR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented. Otherwise, direct accesses to ICC_PPI_SPENDR<n>_EL1 are UNDEFINED.

**Attributes**

ICC_PPI_SPENDR<n>_EL1 is a 64-bit register.

## Field descriptions




**PEND<x> , bits [x], for x = 63 to 0**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Controls the Pending state of PPIs.

Reads return the current state of the INTIDs.

Writing 1 to a field sets the Pending state of the corresponding INTID. Writing 0 has no effect.

| PEND<x> | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Access to this field is RAZ/WI if all of the following are true:
  - !IsPPIAssignedToCurrentDomain((n * 64) + x)
  - PSTATE.EL IN {EL2, EL1}
- When ICC_PPI_HMR_EL1[n].HM<x> == 1, access to this field is RO.
- Otherwise, access to this field is W1S.

## Accessing ICC_PPI_SPENDR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_SPENDR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PENDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_SPENDR_EL1[n];
    else
        X[t, 64] = ICC_PPI_SPENDR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_SPENDR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_SPENDR_EL1[n];
```

**MSR ICC_PPI_SPENDR<n>_EL1, <Xt> ; Where n = 0-1**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_PENDRn_EL1 == '    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_SPENDR_EL1[n] = X[t, 64];
    else
        ICC_PPI_SPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_SPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_SPENDR_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.5 Virtual PPI registers

IQCYQL

Each ICC_PPI_ System register which is accessible at EL1 has a corresponding virtual ICV_PPI_ register. The ICV_PPI_ registers are accessed using the same System register encodings as their ICC_PPI_ counterparts.

Configuration and state of virtual PPIs.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.5.1 ICV_PPI_CACTIVER<n>_EL1, Interrupt Controller Virtual PPI Clear Active Registers, n = 0 - 1

The ICV_PPI_CACTIVER<n>_EL1 characteristics are:

**Purpose**

Clear Active state for virtual PPIs.

**Configuration**

AArch64 System register ICV_PPI_CACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_ACTIVER<n>_EL2[63:0].

AArch64 System register ICV_PPI_CACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_SACTIVER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_CACTIVER<n>_EL1 are UNDEFINED.

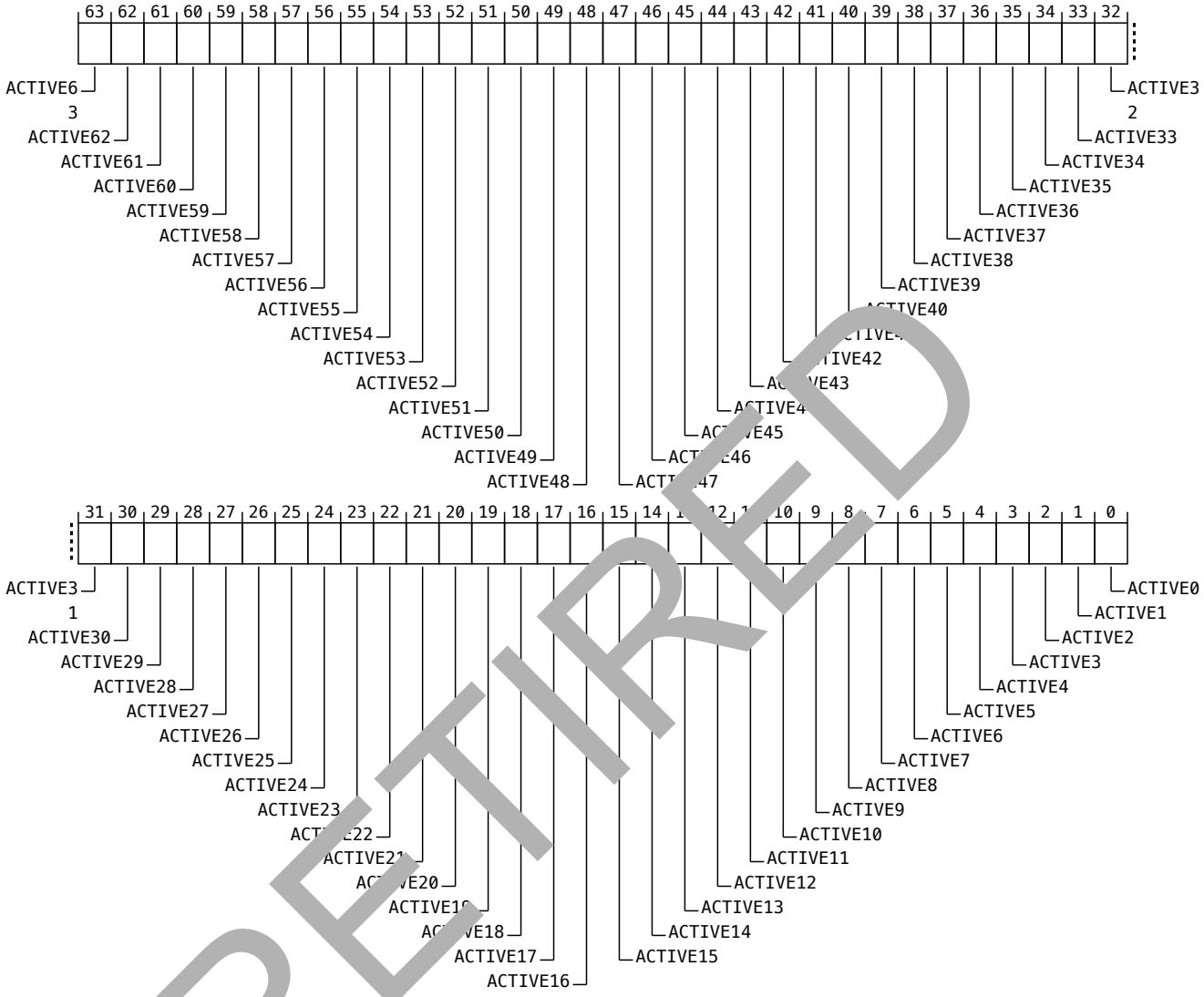
**Attributes**

ICV_PPI_CACTIVER<n>_EL1 is a 64-bit register.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Field descriptions




**ACTIVE<x>, bits [x], for x = 63 to 0**

Configures whether PPIs are Active.

Reads return the Active state of the INTID.

Writing 1 clears the Active state of the INTID. Writing 0 has no effect.

| ACTIVE<x> | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is W1C.

### Accessing ICV_PPI_CACTIVER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_CACTIVER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b00:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeat  Imple nted(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.IC PPI_A TIVERn_ 1 = '0' then
        AArch64.SystemAccessTrap(EL2, 0x1 );
    elsif EL2Enabled() && HCR_EL2.IMO = '1' ther
        X[t, 64] = ICV_PPI_CACTIVER_EL1[n
    else
        X[t, 64] = ICC_PPI_CACTIVER_E [n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_CACTI R_EL1[n];
elsif PSTATE.EL == EL3 the
    X[t, 64] = ICC_PPI_C TIVER 1[n];
```

**MSR ICC_PPI CACTIVER<n> L1, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0 00 | 0b1100 | 0b1101 | 0b00:n[0] |

```
integer  = UInt(op2<0>);

if !IsFeatur mplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_CACTIVER_EL1[n] = X[t, 64];
    else
        ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_CACTIVER_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.5.2 ICV_PPI_CPENDR<n>_EL1, Interrupt Controller Virtual PPI Clear Pending State Registers, n = 0 - 1

The ICV_PPI_CPENDR<n>_EL1 characteristics are:

**Purpose**

Clear pending state for virtual PPIs.

**Configuration**

AArch64 System register ICV_PPI_CPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_PENDR<n>_EL2[63:0].

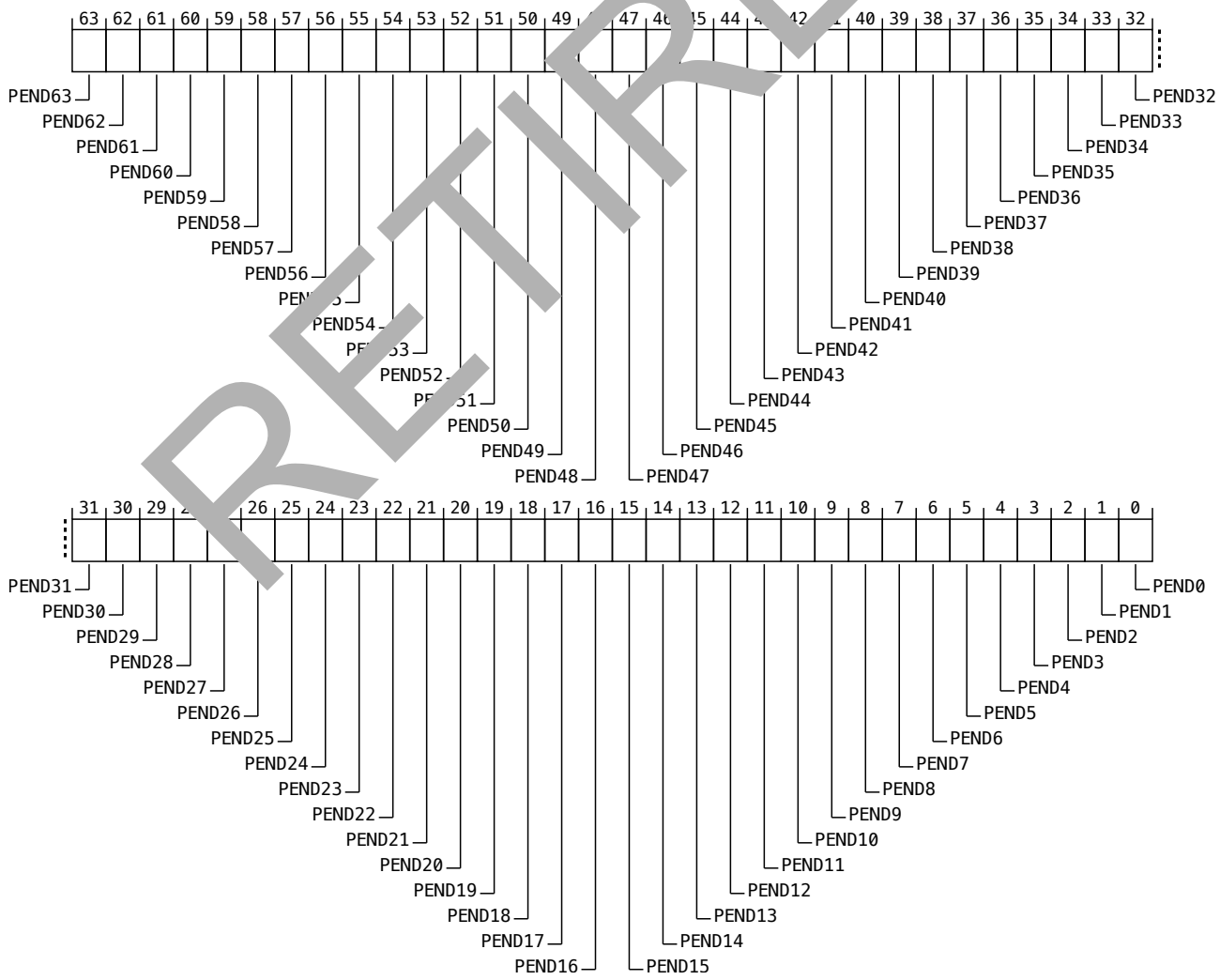
AArch64 System register ICV_PPI_CPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_SPENDR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_CPENDR<n>_EL1 are UNDEFINED.

**Attributes**

ICV_PPI_CPENDR<n>_EL1 is a 64-bit register.

## Field descriptions




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**PEND<x> , bits [x], for x = 63 to 0**

Controls the Pending state of PPIs.

Reads return the current state of the INTIDs.

Writing 1 to a field clears the Pending state of the corresponding INTID. Writing 0 has no effect.

| PEND<x> | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

When the Pending state of a physical PPI is directly injected to the Pending state of virtual PPI <x>, all of the following are true:

- Reads of Pend<x> return the value of the field corresponding to the physical PPI in ICC_PPI_CPENDR<n>_EL1.
- Writes to Pend<x> have the same effect as writes to the field corresponding to the physical PPI in ICC_PPI_CPENDR<n>_EL1.

Otherwise, all of the following are true:

- Reads of Pend<x> return the value of ICH_PPI_PENDR<n>_EL2.Pend<x>.
- Writes to Pend<x> update the value of ICH_PPI_PENDR<n>_EL2.Pend<x>.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- When ICH_PPI_HMR_EL1[n].HM<x> == 1, access to this field is RO.
- Otherwise, access to this field is W1C.

## Accessing ICV_PPI_CPENDR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICV_PPI_CPENDR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PENDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        X[t, 64] = ICV_PPI_CPENDR_EL1[n];
    else
        X[t, 64] = ICC_PPI_CPENDR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_CPENDR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_CPENDR_EL1[n];
```

## MSR ICC_PPI_CPENDR<n>_EL1, <Xt> ; Where n = 0-1

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeature[illegible]emented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2[illegible]CC_PPI_[illegible]NDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x[illegible];
    elsif EL2Enabled() && HCR_EL2.IMO == [illegible] then
        ICV_PPI_CPENDR_EL1[n] = X[t, [illegible]4];
    else
        ICC_PPI_CPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_CPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_CPENDR_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.5.3 ICV_PPI_ENABLER<n>_EL1, Interrupt Controller Virtual PPI Clear Enable Registers, n = 0 - 1

The ICV_PPI_ENABLER<n>_EL1 characteristics are:

**Purpose**

Access to Enabled state for virtual PPIs.

**Configuration**

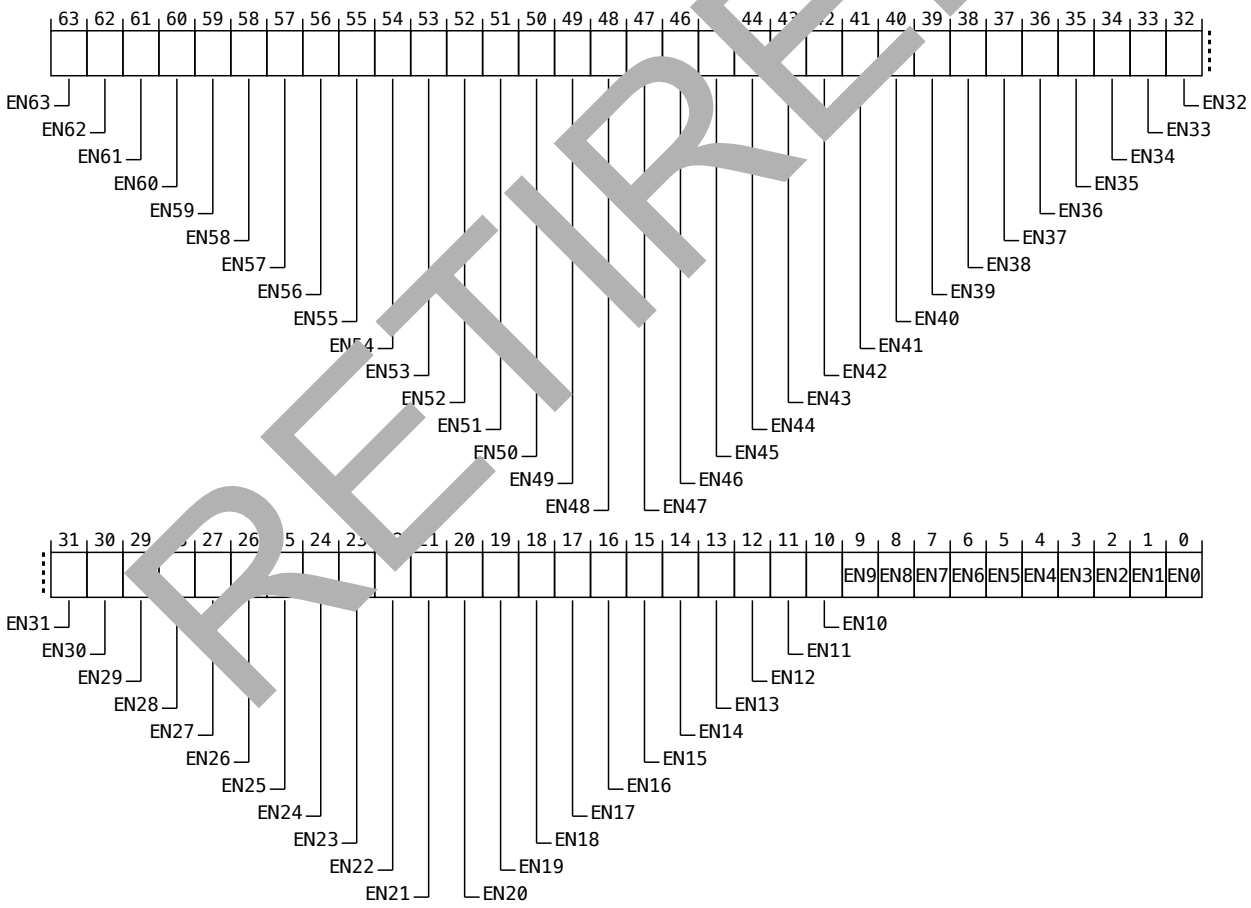
AArch64 System register ICV_PPI_ENABLER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_ENABLER<n>_EL2[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_ENABLER<n>_EL1 are UNDEFINED.

**Attributes**

ICV_PPI_ENABLER<n>_EL1 is a 64-bit register.

## Field descriptions




**EN<x> , bits [x], for x = 63 to 0**

Configures whether PPIs are enabled.

Reads return the current state of the INTID.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| EN<x> | Meaning |
|---|---|
| 0b0 | Disabled |
| 0b1 | Enabled |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RW.

### Accessing ICV_PPI_ENABLER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_ENABLER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b101[illegible] | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HCR_EL2.ICC_PPI_ENABLERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_ENABLER_EL1[n];
    else
        X[t, 64] = ICC_PPI_ENABLER_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_ENABLER_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_ENABLER_EL1[n];
```

**MSR ICC_PPI_ENABLER<n>_EL1, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ENABLERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_ENABLER_EL1[n] = X[t, 64];
    else
        ICC_PPI_ENABLER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_ENABLER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_ENABLER_EL1[n] = X[t, 64];
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.5.4 ICV_PPI_HMR<n>_EL1, Interrupt Controller Virtual PPI Handling mode Registers, n = 0 - 1

The ICV_PPI_HMR<n>_EL1 characteristics are:

**Purpose**

Report whether virtual PPIs are Edge or Level.

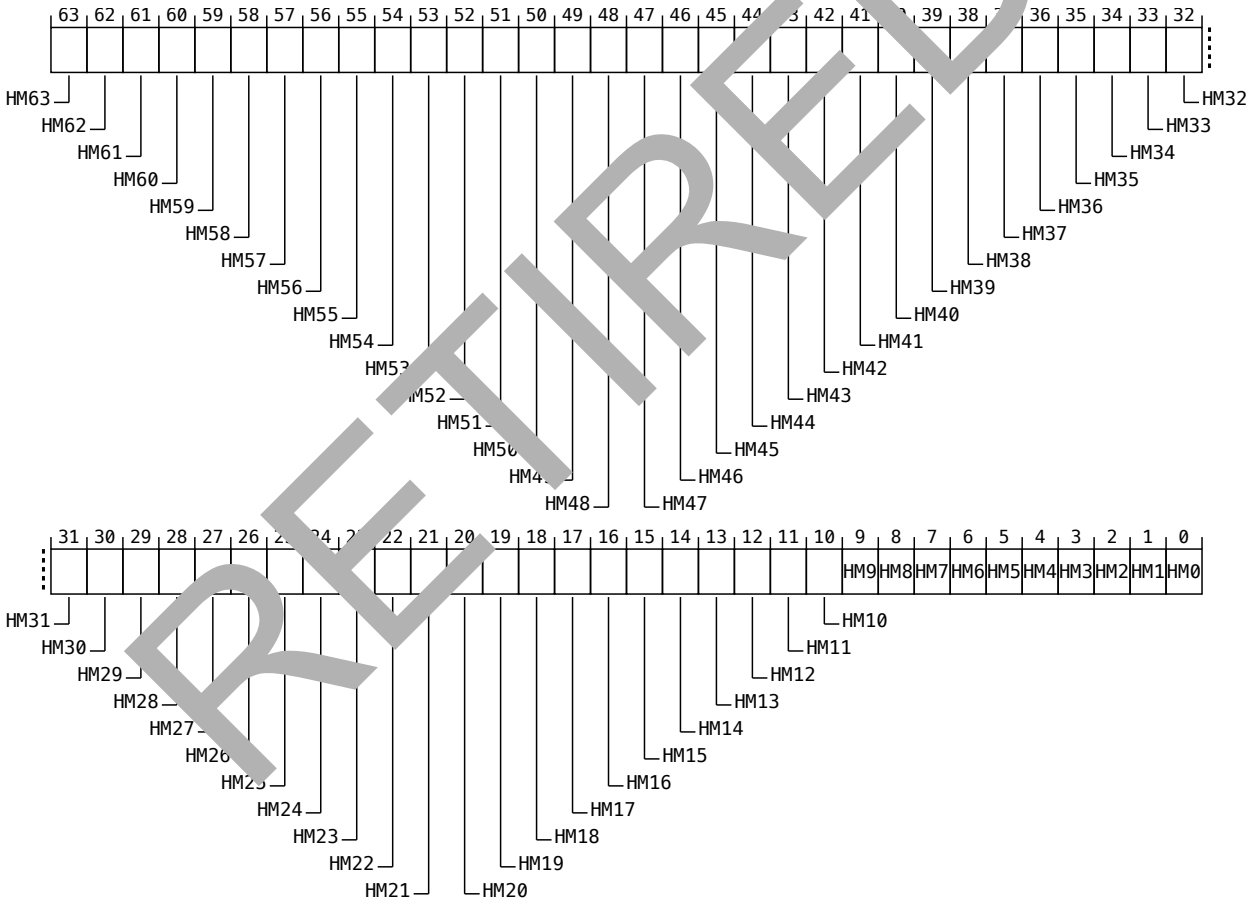
**Configuration**

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_HMR<n>_EL1 are UNDEFINED.

**Attributes**

ICV_PPI_HMR<n>_EL1 is a 64-bit register.

## Field descriptions




**HM<x> , bits [x], for x = 63 to 0**

The PPI Handling mode.

| HM<x> | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | Level |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RO.

### Accessing ICV_PPI_HMR<n>_EL1

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_HMR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1010 | 00:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_HMRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_HMR_EL1[n];
    else
        X[t, 64] = ICC_PPI_HMR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_HMR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_HMR_EL1[n];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.5.5 ICV_PPI_PRIORITYR<n>_EL1, Interrupt Controller Virtual PPI Priority Registers, n = 0 - 15

The ICV_PPI_PRIORITYR<n>_EL1 characteristics are:

**Purpose**

Configures the priority of virtual PPIs.

**Configuration**

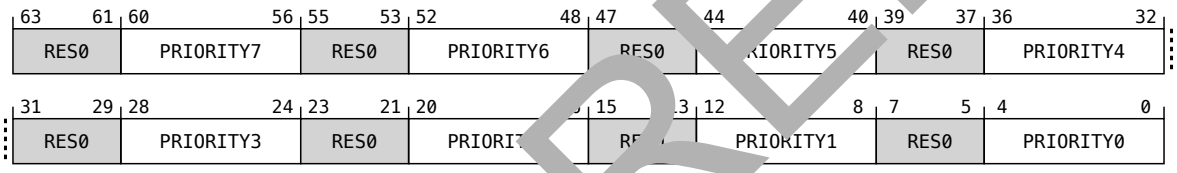
AArch64 System register ICV_PPI_PRIORITYR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_PRIORITYR<n>_EL2[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_PRIORITYR<n>_EL1 are UNDEFINED.

**Attributes**

ICV_PPI_PRIORITYR<n>_EL1 is a 64-bit register.

## Field descriptions




**Bits [63:61, 55:53, 47:45, 39:37, 31:29, 23:21, 15:13, 7:5]**

Reserved, RES0.

**PRIORITY<x> , bits [60:56, 52:48, 44:40, 36:32, 28:24, 20:16, 12:8, 4:0], for x = 7 to 0**

Configures the priority of the corresponding PPI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IsPPIImplemented((n * 8) + x), access to this field is RES0.
  Otherwise, access to this field is RW.

## Accessing ICV_PPI_PRIORITYR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_PRIORITYR<n>_EL1 ; Where n = 0-15**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b111:n[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PRIORITYRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PPI_PRIORITYR_EL1[n];
    else
        X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_PRIORITYR_EL1[n];
```

## MSR ICC_PPI_PRIORITYR<n>_EL1, <Xt> ; Where = 0-15

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b[illegible]1:n[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.[illegible] == [illegible] && [illegible]atureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == [illegible]1' then
        UNDEFINED;
    elsif EL2Enabled() && [illegible]CH_[illegible]WTR_EL2.ICC_[illegible]I_PRIORITYRn_EL1 == '0' then
        AArch64.SystemAccessTrap[illegible]2, 0x18);
    elsif EL2Enabled([illegible] && HCR_EL2.[illegible] == '1' then
        ICV_PPI_PR[illegible]ITYR_EL1[n] = X[illegible]64];
    else
        ICC_PP[illegible]RIORITY[illegible]EL1[n] = X[t, 64];
elsif PSTATE.EL [illegible] EL2[illegible]hen
    ICC_PPI_PRIOR[illegible]EL1[n] = [illegible]t, 64];
elsif PS[illegible] == E[illegible] then
    IC[illegible]PPI_PR[illegible]ORITYR_[illegible] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.5.6 ICV_PPI_SACTIVER<n>_EL1, Interrupt Controller Virtual PPI Set Active Registers, n = 0 - 1

The ICV_PPI_SACTIVER<n>_EL1 characteristics are:

**Purpose**

Set Active state for virtual PPIs.

**Configuration**

AArch64 System register ICV_PPI_SACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_ACTIVER<n>_EL2[63:0].

AArch64 System register ICV_PPI_SACTIVER<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_CACTIVER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_SACTIVER<n>_EL1 are UNDEFINED.

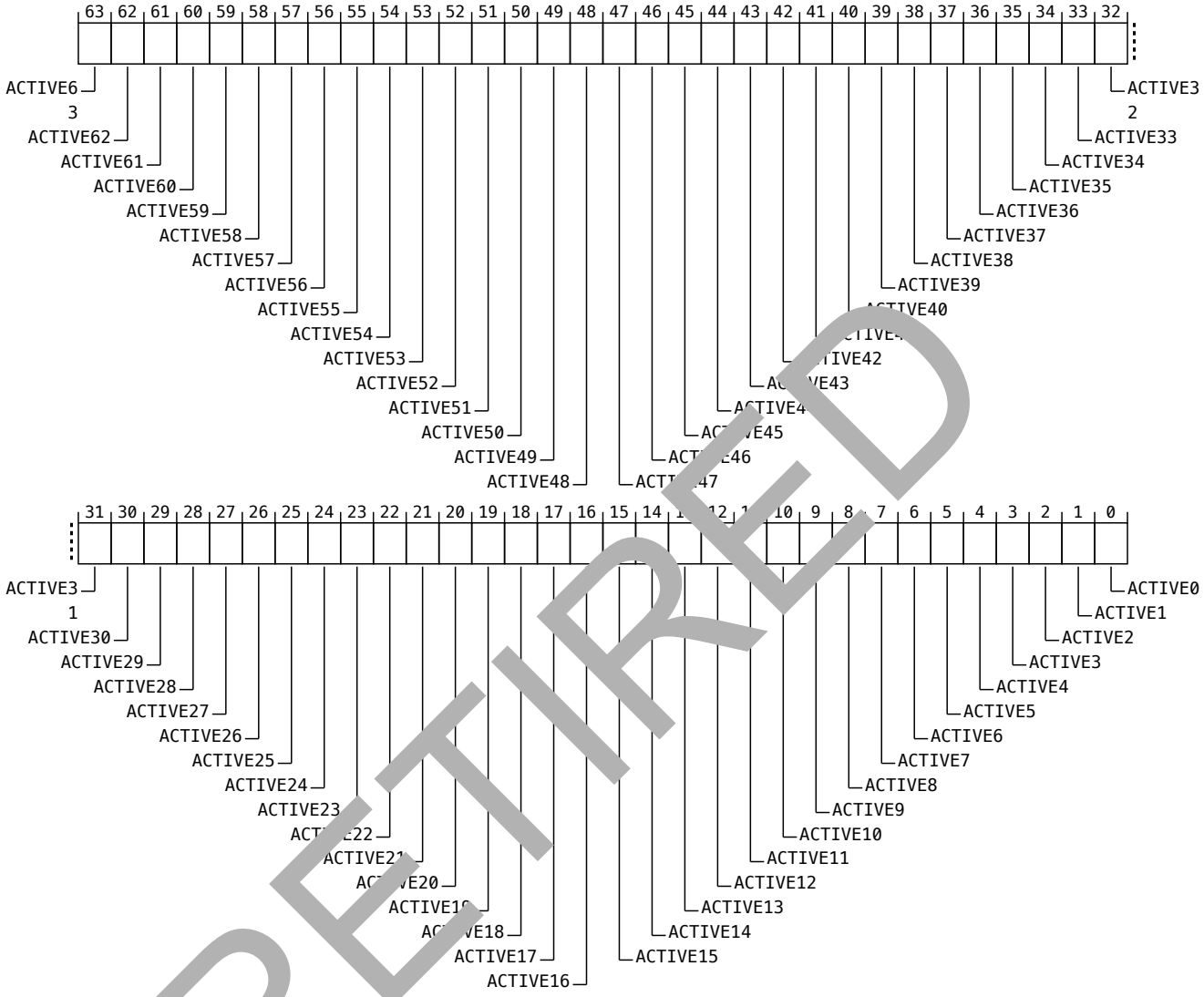
**Attributes**

ICV_PPI_SACTIVER<n>_EL1 is a 64-bit register.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Field descriptions




**ACTIVE<x>, bits [x], for x = 63 to 0**

Configures whether PPIs are Active.

Reads return the Active state of the INTID.

Writing 1 sets the Active state of the INTID. Writing 0 has no effect.

| ACTIVE<x> | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is W1S.

### Accessing ICV_PPI_SACTIVER<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICC_PPI_SACTIVER<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b01:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeat  Imple nted(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.IC PPI_A TIVERn_ 1 = '0' then
        AArch64.SystemAccessTrap(EL2, 0x1 );
    elsif EL2Enabled() && HCR_EL2.IMO = '1' ther
        X[t, 64] = ICV_PPI_SACTIVER_EL1[n];
    else
        X[t, 64] = ICC_PPI_SACTIVER_E [n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_SACTI R_EL1[n];
elsif PSTATE.EL == EL3 the
    X[t, 64] = ICC_PPI_S TIVER 1[n];
```

**MSR ICC_PPI SACTIVER<n> L1, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0 00 | 0b1100 | 0b1101 | 0b01:n[0] |

```
integer  = UInt(op2<0>);

if !IsFeatur mplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2.ICC_PPI_ACTIVERn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PPI_SACTIVER_EL1[n] = X[t, 64];
    else
        ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_SACTIVER_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.5.7 ICV_PPI_SPENDR<n>_EL1, Interrupt Controller Virtual PPI Set Pending State Registers, n = 0 - 1

The ICV_PPI_SPENDR<n>_EL1 characteristics are:

**Purpose**

Set pending state for virtual PPIs.

**Configuration**

AArch64 System register ICV_PPI_SPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICH_PPI_PENDR<n>_EL2[63:0].

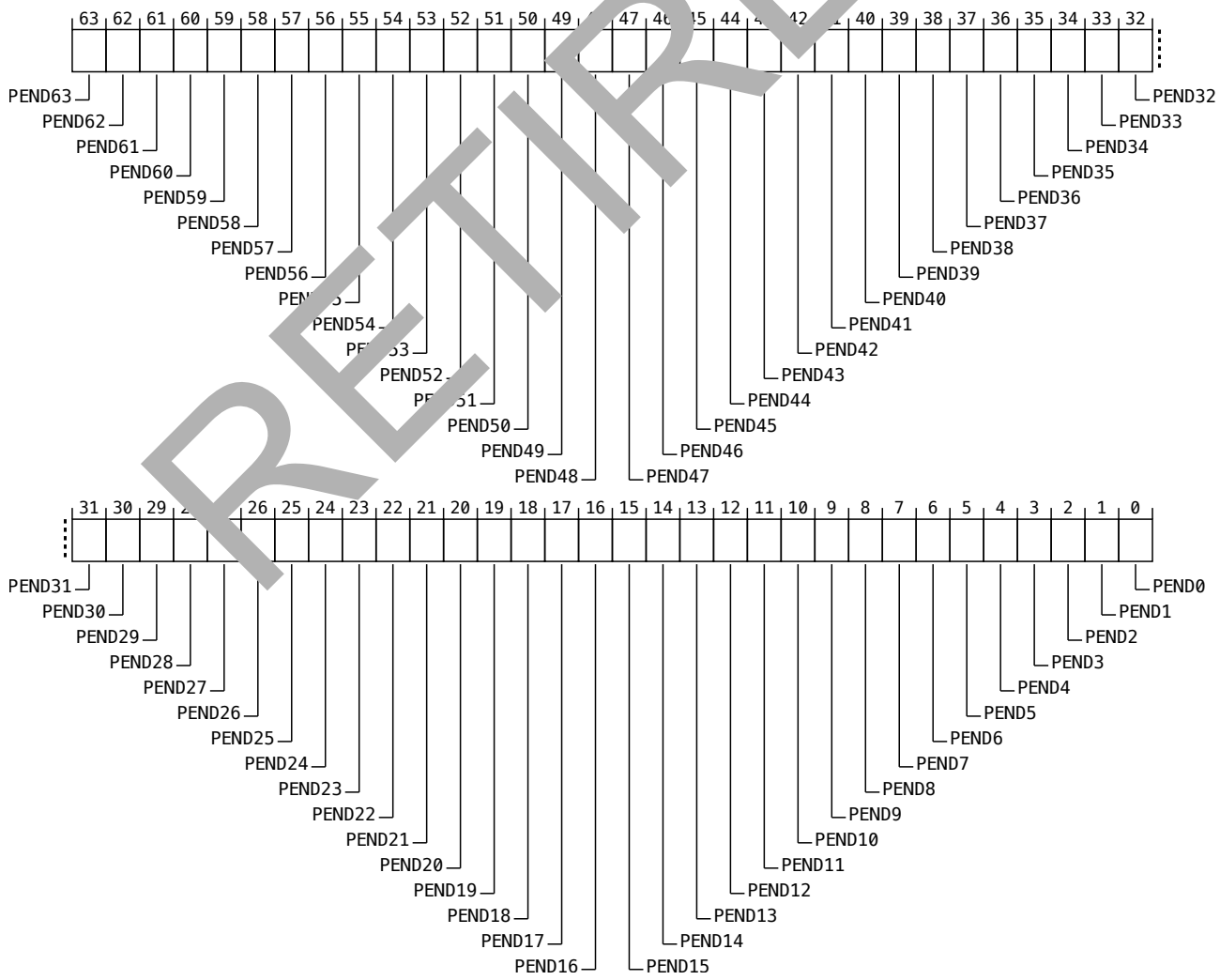
AArch64 System register ICV_PPI_SPENDR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_CPENDR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PPI_SPENDR<n>_EL1 are UNDEFINED.

**Attributes**

ICV_PPI_SPENDR<n>_EL1 is a 64-bit register.

## Field descriptions




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**PEND<x> , bits [x], for x = 63 to 0**

Controls the Pending state of PPIs.

Reads return the current state of the INTIDs.

Writing 1 to a field sets the Pending state of the corresponding INTID. Writing 0 has no effect.

| PEND<x> | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

When the Pending state of a physical PPI is directly injected to the Pending state of virtual PPI <x>, all of the following are true:

- Reads of Pend<x> return the value of the field corresponding to the physical PPI in ICC_PPI_SPENDR<n>_EL1.
- Writes to Pend<x> have the same effect as writes to the field corresponding to the physical PPI in ICC_PPI_SPENDR<n>_EL1.

Otherwise, all of the following are true:

- Reads of Pend<x> return the value of ICH_PPI_PENDR<n>_EL2.Pend<x>.
- Writes to Pend<x> update the value of ICH_PPI_PENDR<n>_EL2.Pend<x>.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- When ICH_PPI_HMR_EL1[n].HM<x> == 1, access to this field is RO.
- Otherwise, access to this field is W1S.

## Accessing ICV_PPI_SPENDR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICV_PPI_SPENDR<n>_EL1 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeatureImplemented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGRTR_EL2.ICC_PPI_PENDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        X[t, 64] = ICV_PPI_SPENDR_EL1[n];
    else
        X[t, 64] = ICC_PPI_SPENDR_EL1[n];
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICC_PPI_SPENDR_EL1[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_PPI_SPENDR_EL1[n];
```

## MSR ICC_PPI_SPENDR<n>_EL1, <Xt> ; Where n = 0-1

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b000 | 0b1100 | 0b1101 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !IsFeatureImplemented(FEAT_GCIE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.IMO == '1' && IsFeature[illegible]emented(FEAT_GCIE_LEGACY) &&
        ↪ICH_VCTLR_EL2.V3 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ICH_HFGWTR_EL2[illegible]CC_PPI_[illegible]NDRn_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x[illegible];
    elsif EL2Enabled() && HCR_EL2.IMO == [illegible]t[illegible]n
        ICV_PPI_SPENDR_EL1[n] = X[t, [illegible]4];
    else
        ICC_PPI_SPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL2 then
    ICC_PPI_SPENDR_EL1[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICC_PPI_SPENDR_EL1[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.6 Hypervisor control registers

Registers to manage operation of the GICv5 and Legacy virtual CPU interface.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.6.1 ICH_APR_EL2, Interrupt Controller Active Virtual Priorities Register

The ICH_APR_EL2 characteristics are:

**Purpose**

Records active priorities for the Virtual Interrupt Domain.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

When FEAT_GCIE_LEGACY is implemented, AArch64 System register ICH_APR_EL2 bits [31:0] are architecturally mapped to AArch64 System register ICH_AP1R0_EL2[31:0].

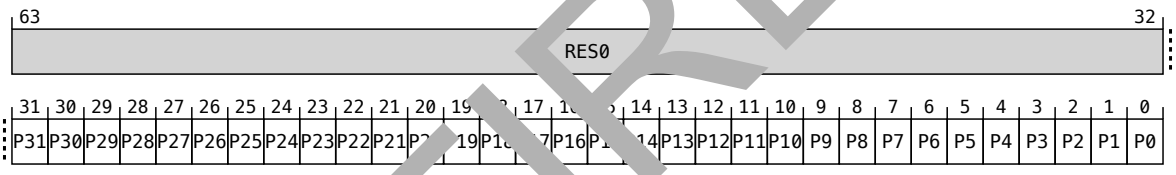
AArch64 System register ICH_APR_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_APR_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_APR_EL2 are UNDEFINED.

**Attributes**

ICH_APR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**P<x> , bits [x], for x = 31 to 0**

Provides access to the active virtual priorities.

| P<x> | Meaning |
|---|---|
| 0b0 | Priority not active |
| 0b1 | Priority active |

This field is an alias of the equivalent field in ICV_APR_EL1.

## Accessing ICH_APR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_APR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b100 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB00];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_APR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_APR_EL2;
```

## MSR ICH_APR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b100 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB00] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_APR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_APR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.2 ICH_CONTEXTR_EL2, Interrupt Controller Virtual Context Register

The ICH_CONTEXTR_EL2 characteristics are:

**Purpose**

Selects the current resident VPE.

**Configuration**

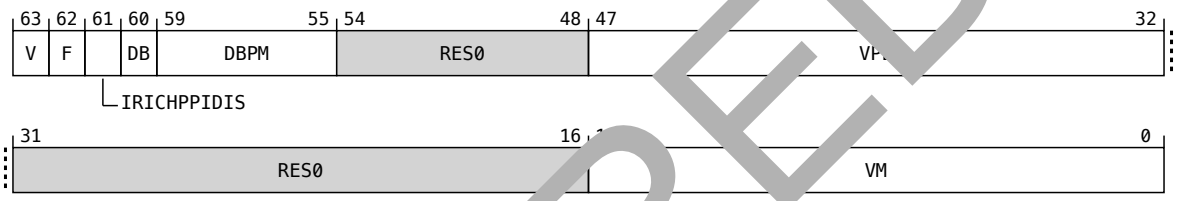
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_CONTEXTR_EL2 are UNDEFINED.

**Attributes**

ICH_CONTEXTR_EL2 is a 64-bit register.

## Field descriptions




**V, bit [63]**

Indicates whether a VPE is currently resident.

If EL2 is not enabled in the current security state identified by SCR_EL3.{NSE,NS}, the Effective value of this field is 0.

| V | Meaning |
|---|---|
| 0b0 | No VPE is resident. |
| 0b1 | A VPE is resident. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**F, bit [62]**

Indicates whether the last write that set V to 1 succeeded in making a VPE resident.

| F | Meaning |
|---|---|
| 0b0 | Success |
| 0b1 | Fault |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**IRICHPPIDIS, bit [61]**

Specifies whether the candidate HPPI presented by the IRI for the resident VPE is considered when selecting the HPPI for the Virtual Interrupt Domain.

When V is 0, this field is IGNORED.

| IRICHPPIDIS | Meaning |
|---|---|
| 0b0 | The CPU interface considers both the virtual PPIs and the candidate HPPI presented by the IRI when determining the HPPI for the Virtual Interrupt Domain. |
| 0b1 | The CPU interface only considers the virtual PPIs when determining the HPPI for the Virtual Interrupt Domain. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**DB, bit [60]**

Specifies whether a doorbell interrupt is requested when making a VPE non-resident.

| DB | Meaning |
|---|---|
| 0b0 | No doorbell requested. |
| 0b1 | Doorbell requested. |

On a write that changes V from 1 to 0, this field specifies whether a doorbell interrupt is requested for the previously resident VPE.

For all other writes, this field is IGNORED.

On reads, this field is RAZ.

If the current value of SCR_EL3.{NSE,NS} is different from the value at the time the VPE was made resident, this field behaves as if set to 0.

This field has no effect if EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**DBPM, bits [59:55]**

Doorbell priority mask.

On a write that changes V from 1 to 0, when DB is written as 1, this field specifies the minimum priority for a virtual interrupt to trigger the VPE's doorbell.

For all other writes, this field is IGNORED.

On reads, this field is RAZ.

This field has no effect if EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [54:48]**

Reserved, RES0.

**VPE, bits [47:32]**

When V is 1, identifies the resident VPE.

When V is 0, this field is IGNORED.

This field has no effect if EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [31:16]**

Reserved, RES0.

**VM, bits [15:0]**

When V is 1, identifies the resident VM.

When V is 0, this field is IGNORED.

This field has no effect if EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICH_CONTEXTR_EL2

Accessions to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_CONTEXTR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b110 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB08];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_CONTEXTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_CONTEXTR_EL2;
```

**MSR ICH_CONTEXTR_EL2, <Xt>**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b110 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB08] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2        '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_CONTEXTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_CONTEXTR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.3 ICH_HFGITR_EL2, Hypervisor GIC Fine-Grained Instruction Trap Register

The ICH_HFGITR_EL2 characteristics are:

**Purpose**

Provides instruction trap controls for GIC System instructions.

**Configuration**

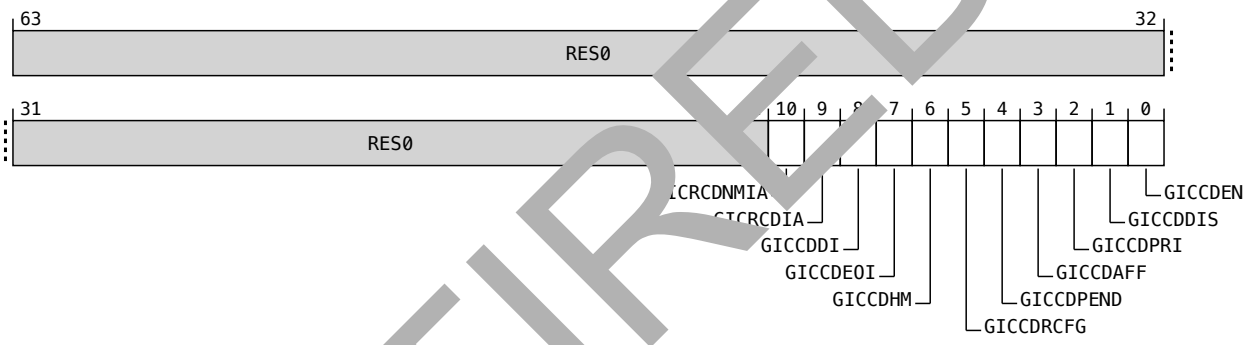
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HFGITR_EL2 are UNDEFINED.

**Attributes**

ICH_HFGITR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:11]**

Reserved, RES0.

**GICRCDNMIA, bit [10]**

Trap execution of GICR CDNMIA at EL1 to EL2.

| GICRCDNMIA | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GICR CDNMIA at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GICR CDNMIA is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICRCDIA, bit [9]**

Trap execution of GICR CDIA at EL1 to EL2.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| GICRCDIA | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GICR CDIA at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GICR CDIA is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDDI, bit [8]**

Trap execution of GIC CDDI at EL1 to EL2.

| GICCDDI | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDDI at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDDI is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDEOI, bit [7]**

Trap execution of GIC CDEOI at EL1 to EL2.

| GICCDEOI | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDEOI at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDEOI is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDHM, bit [6]**

Trap execution of GIC CDHM at EL1 to EL2.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| GICCDHM | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDHM at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDHM is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDRCFG, bit [5]**

Trap execution of GIC CDRCFG at EL1 to EL2.

| GICCDRCFG | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDRCFG at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDRCFG is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDPEND, bit [4]**

Trap execution of GIC CDPEND at EL1 to EL2.

| GICCDPEND | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDPEND at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDPEND is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDAFF, bit [3]**

Trap execution of GIC CDAFF at EL1 to EL2.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| GICCDAFF | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDAFF at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDAFF is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDPRI, bit [2]**

Trap execution of GIC CDPRI at EL1 to EL2.

| GICCDPRI | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDPRI at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDPRI is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDDIS, bit [1]**

Trap execution of GIC CDDIS at EL1 to EL2.

| GICCDDIS | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDDIS at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDDIS is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**GICCDEN, bit [0]**

Trap execution of GIC CDEN at EL1 to EL2.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| GICCDEN | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then execution of GIC CDEN at EL1 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of GIC CDEN is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICH_HFGITR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_HFGITR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b100 | 0b111 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB10];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_HFGITR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_HFGITR_EL2;
```

**MSR ICH_HFGITR_EL2, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b111 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB10] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_HFGITR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_HFGITR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.4 ICH_HFGRTR_EL2, Hypervisor GIC Fine-Grained Read Trap Register

The ICH_HFGRTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MRS reads of GIC System registers.

**Configuration**

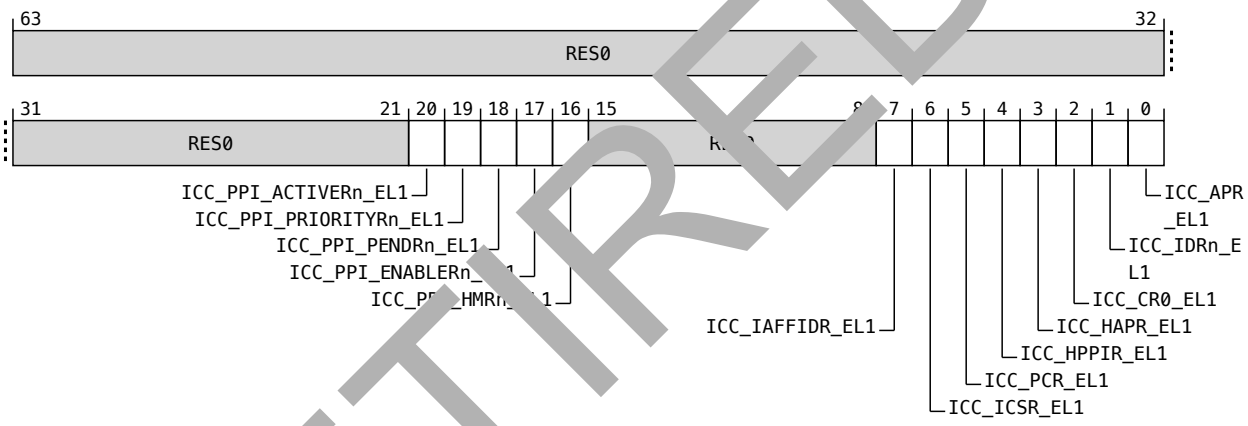
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HFGRTR_EL2 are UNDEFINED.

**Attributes**

ICH_HFGRTR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:21]**

Reserved, RES0.

**ICC_PPI_ACTIVERn_EL1, bits [20]**

Trap MRS reads of ICC_PPI_CACTIVER<n>_EL1, ICC_PPI_SACTIVER<n>_EL1, ICV_PPI_CACTIVER<n>_EL1 and ICV_PPI_SACTIVER<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_ACTIVERn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**ICC_PPI_PRIORITYRn_EL1, bit [19]**

Trap MRS reads of ICC_PPI_PRIORITYR<n>_EL1 and ICV_PPI_PRIORITYR<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_PRIORITYRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_PENDRn_EL1, bit [18]**

Trap MRS reads of ICC_PPI_PENDR<n>_EL1, ICC_PPI_SPENDR<n>_EL1 ICV_PPI_CPENDR<n>_EL1 and ICV_PPI_SPENDR<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_PENDRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_ENABLERn_EL1, bit [17]**

Trap MRS reads of ICC_PPI_ENABLER<n>_EL1 and ICV_PPI_ENABLER<n>_EL1 at EL1 using AArch64 to EL2.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| ICC_PPI_ENABLERn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_HMRn_EL1, bit [16]**

Trap MRS reads of ICC_PPI_HMR<n>_EL1 and ICV_PPI_HMR<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_HMRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [15:8]**

Reserved, RES0.

**ICC_IAFFIDR_EL1, bit [7]**

Trap MRS reads of ICC_IAFFIDR_EL1 at EL1 using AArch64 to EL2.

| ICC_IAFFIDR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_ICSR_EL1, bit [6]**

Trap MRS reads of ICC_ICSR_EL1 at EL1 using AArch64 to EL2.

| ICC_ICSR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PCR_EL1, bit [5]**

Trap MRS reads of ICC_PCR_EL1 and ICV_PCR_EL1 at EL1 using AArch64 to EL2.

| ICC_PCR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_HPPIR_EL1, bit [4]**

Trap MRS reads of ICC_HPPIR_EL1 and ICV_HPPIR_EL1 at EL1 using AArch64 to EL2.

| ICC_HPPIR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_HAPR_EL1, bit [3]**

Trap MRS reads of ICC_HAPR_EL1 and ICV_HAPR_EL1 at EL1 using AArch64 to EL2.

| ICC_HAPR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_CR0_EL1, bit [2]**

Trap MRS reads of ICC_CR0_EL1 and ICV_CR0_EL1 at EL1 using AArch64 to EL2.

| ICC_CR0_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_IDR0_EL1, bit [1]**

Trap MRS reads of ICC_IDR0_EL1 at EL1 using AArch64 to EL2.

| ICC_IDRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_APR_EL1, bit [0]**

Trap MRS reads of ICC_APR_EL1 and ICV_APR_EL1 at EL1 using AArch64 to EL2.

| ICC_APR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MRS reads of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the s[illegible]isters are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKN[illegible]N value.

## Accessing ICH_HFGRTR_EL2

Accesses to this register use the following enc[illegible]ngs in [illegible]e Syste[illegible]ister encoding space:

**MRS <Xt>, ICH_HFGRTR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b100 |

```
if !(IsFeatureImp[illegible]mented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL [illegible]EL0 [illegible]en
    UNDEFINED;
elsif PST[illegible] == E[illegible] then
    if [illegible]ifecti[illegible]HCR_EL[illegible]NV[illegible] IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(Is[illegible]eatureImp[illegible]mented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t[illegible]NVMem[0xB18];
    el[illegible]f [illegible]rectiveH[illegible]_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        (IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AA[illegible]64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_HFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_HFGRTR_EL2;
```

**MSR ICH_HFGRTR_EL2, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b100 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB18] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_HFGRTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_HFGRTR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.5 ICH_HFGWTR_EL2, Hypervisor GIC Fine-Grained Write Trap Register

The ICH_HFGWTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MSR writes of GIC System registers.

**Configuration**

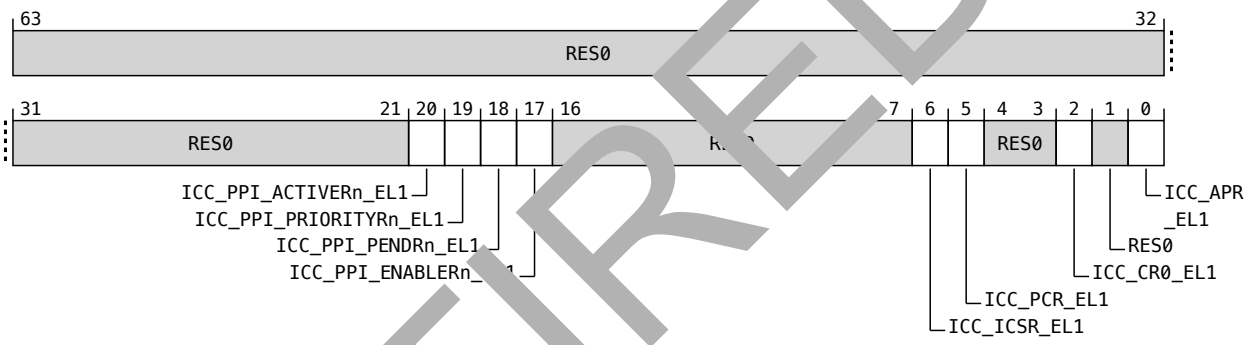
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HFGWTR_EL2 are UNDEFINED.

**Attributes**

ICH_HFGWTR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:21]**

Reserved, RES0.

**ICC_PPI_ACTIVERn_EL1, bit [20]**

Trap MSR writes ICC_PPI_CACTIVER<n>_EL1, ICC_PPI_SACTIVER<n>_EL1, ICV_PPI_CACTIVER<n>_EL1 and ICV_PPI_SACTIVER<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_ACTIVER | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_PRIORITYRn_EL1, bit [19]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Trap MSR writes of ICC_PPI_PRIORITYR<n>_EL1 and ICV_PPI_PRIORITYR<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_PRIORITYRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_PENDRn_EL1, bit [18]**

Trap MSR writes of ICC_PPI_PENDR<n>_EL1, ICC_PPI_SPENDR<n>_EL1 ICV_PPI_CPENDR<n>_EL1 and ICV_PPI_SPENDR<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_PENDRn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PPI_ENABLERn_EL1, bit [17]**

Trap MSR writes of ICC_PPI_ENABLER<n>_EL1 and ICV_PPI_ENABLER<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_PPI_ENABLERn_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| ICC_PPI_ENABLERn_EL1 | Meaning |
|---|---|
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [16:7]**

Reserved, RES0.

**ICC_ICSR_EL1, bit [6]**

Trap MSR writes of ICC_ICSR_EL1 at EL1 using AArch64 to EL2.

| ICC_ICSR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**ICC_PCR_EL1, bit [5]**

Trap MSR writes of ICC_PCR_EL1 and ICV_PCR_EL1 at EL1 using AArch64 to EL2.

| ICC_PCR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bits [4:3]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**ICC_CR0_EL1, bit [2]**

Trap MSR writes of ICC_CR0_EL1 and ICV_CR0_EL1 at EL1 using AArch64 to EL2.

| ICC_CR0_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

**Bit [1]**

Reserved, RES0.

**ICC_APR_EL1, bit [0]**

Trap MSR writes of ICC_APR_EL1 and ICV_APR_EL1 at EL1 using AArch64 to EL2.

| ICC_APR_EL1 | Meaning |
|---|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, then MSR writes of the specified registers at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MSR writes of the specified registers are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

## Accessing ICH_HFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_HFGWTR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b110 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB20];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_HFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_HFGWTR_EL2;
```

## MSR ICH_HFGWTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b10[illegible] | 0b110 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL[illegible] || Have[illegible](EL3))) t[illegible]n
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} [illegible]& (!IsFe[illegible]ureImple[illegible]nted(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GC[illegible]LEGAC[illegible]VCTLR_EL2.V3 == '0')) then
        NVMem[0xB20] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN [illegible]xx1'} [illegible](!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented[illegible]EAT_[illegible]E_LEGA[illegible]) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessT[illegible](EL2, 0x[illegible];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_HFGWTR_EL2 = [illegible]t, 64];
elsif PSTATE.EL == [illegible]3 then
    ICH_HFGWTR_EL[illegible] = X[t, [illegible]4];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.6 ICH_HPPIR_EL2, Interrupt Controller Hypervisor Highest Priority Pending Interrupt Register

The ICH_HPPIR_EL2 characteristics are:

**Purpose**

Reports the HPPI for the Virtual Interrupt Domain.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

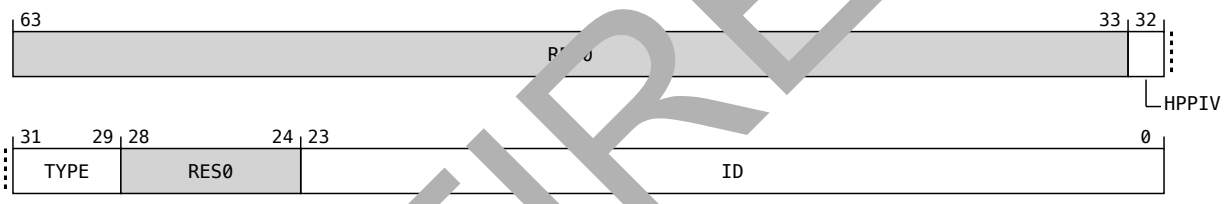
AArch64 System register ICH_HPPIR_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_HPPIR_EL1[63:0].

This register is present only when FEAT_GCIE is implemented (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HPPIR_EL2 are UNDEFINED.

**Attributes**

ICH_HPPIR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:33]**

Reserved, RES0.

**HPPIV, bit [32]**

HPPI valid.

The ... HPPI ... ith Sufficient priority for the Interrupt Domain.

| HPPIV | Meaning |
|---|---|
| 0b0 | Invalid: There is no HPPI with Sufficient priority for the Interrupt Domain. |
| 0b1 | VALID: There is an HPPI with Sufficient priority for the Interrupt Domain. |

If ICH_HPPIR_EL2.HPPIV is 1, ID and TYPE together form the INTID of the HPPI for the Interrupt Domain.

**TYPE, bits [31:29]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICH_HPPIR_EL2.HPPIV is 0, TYPE is RES0.
- If ICH_HPPIR_EL2.HPPIV is 1, TYPE specifies the Type of the interrupt.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| TYPE | Meaning |
|---|---|
| 0b001 | PPI |
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The encoding of this field depends on the value of HPPIV as described below:

- If ICH_HPPIR_EL2.HPPIV is 0, ID is RES0.
- If ICH_HPPIR_EL2.HPPIV is 1, ID specifies the interrupt ID.

## Accessing ICH_HPPIR_EL2

This register is Read-only.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_HPPIR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b101 |

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NIT() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
      ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_HPPIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_HPPIR_EL2;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.7 ICH_PPI_ACTIVER<n>_EL2, Interrupt Controller Virtual Interrupt Active Registers, n = 0 - 1

The ICH_PPI_ACTIVER<n>_EL2 characteristics are:

**Purpose**

Access to Active state for virtual PPIs.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register ICH_PPI_ACTIVER<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_CACTIVER<n>_EL1[63:0].

AArch64 System register ICH_PPI_ACTIVER<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_SACTIVER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_PPI_ACTIVER<n>_EL2 are UNDEFINED.

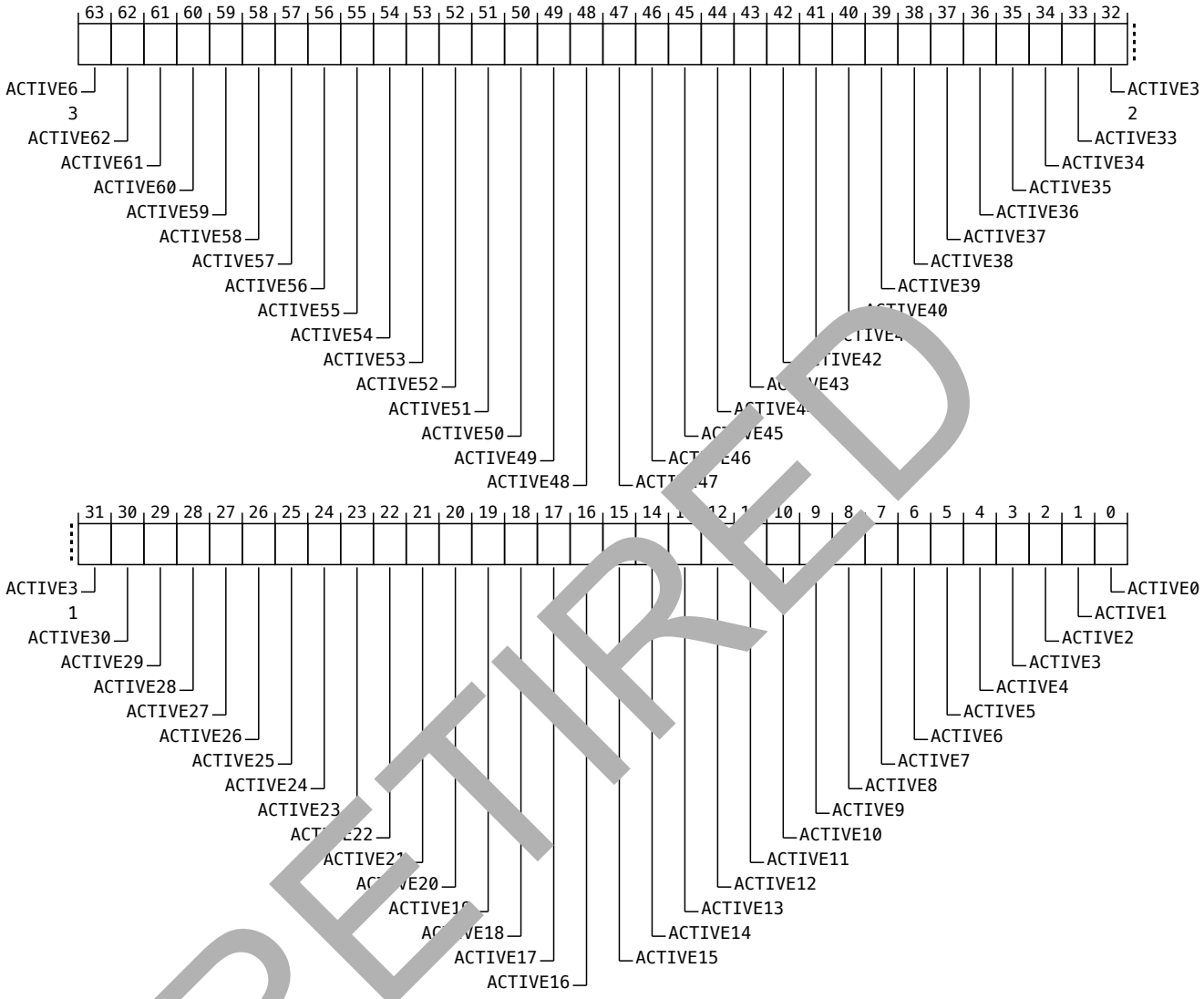
**Attributes**

ICH_PPI_ACTIVER<n>_EL2 is a 64-bit register.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Field descriptions




**ACTIVE<x>, bits [x], for x = 63 to 0**

Access the Active state of the virtual PPI.

| ACTIVE<x> | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Accessing ICH_PPI_ACTIVER<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_PPI_ACTIVER<n>_EL2 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB30 + (8 * n)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_PPI_ACTIVER_EL2[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_PPI_ACTIVER_EL2[n];
```

**MSR ICH_PPI_ACTIVER<n>_EL2, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b11:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB30 + (8 * n)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_PPI_ACTIVER_EL2[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_PPI_ACTIVER_EL2[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.8 ICH_PPI_DVIR<n>_EL2, Interrupt Controller PPI Direct-inject Virtual Interrupt Registers, n = 0 - 1

The ICH_PPI_DVIR<n>_EL2 characteristics are:

**Purpose**

Controls whether Pending physical PPIs are directly injected as virtual PPIs to the Virtual Interrupt Domain.

**Configuration**

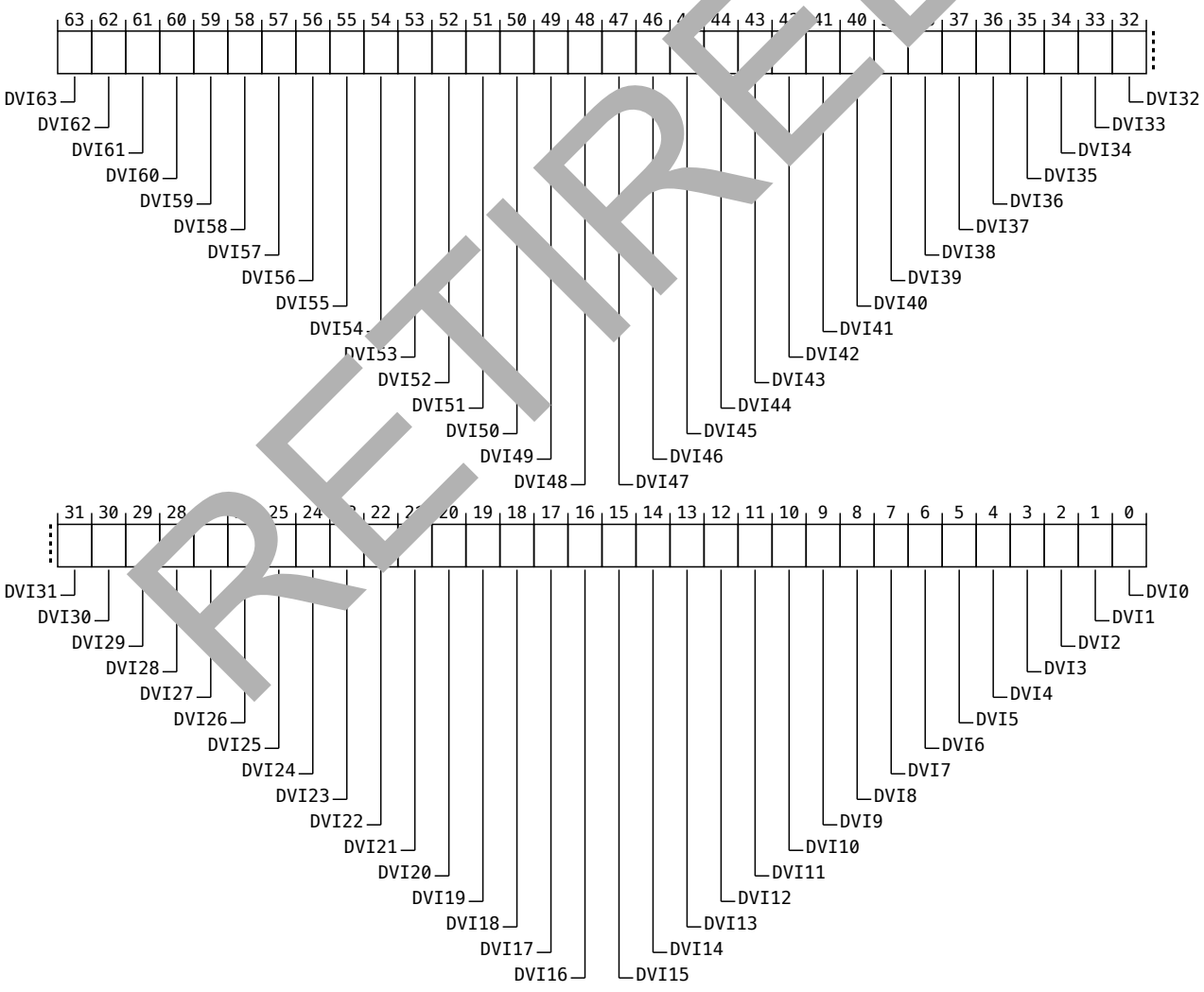
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_PPI_DVIR<n>_EL2 are UNDEFINED.

**Attributes**

ICH_PPI_DVIR<n>_EL2 is a 64-bit register.

## Field descriptions




**DVI<x> , bits [x], for x = 63 to 0**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Controls whether the physical PPI Pending state is directly injected to a virtual PPI in the Virtual Interrupt Domain.

| DVI<x> | Meaning |
|---|---|
| 0b0 | Physical PPI <x> is not directly injected as a virtual PPI. |
| 0b1 | Physical PPI <x> is directly injected as a virtual PPI. |

The Effective value of DVI<x> is 0 if any of the following are true:

- Physical PPI <x> is not assigned to Current Physical Interrupt D
- Physical PPI <x> is assigned to the EL3 Interrupt Domain.
- Physical PPI <x> is assigned to the Secure Interrupt Domai d SCR_EL3 EL2 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNO N value.

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + , acce to this ld i RES0.
- Access to this field is RAZ/WI if a of the fo owing are t e:
  - !IsPPIAssignedToCurrentDom ((n *
  - PSTATE.EL IN {EL2, E }
- Otherwise, access to this eld is W.

### Accessing ICH_PPI_DV <n>_EL2

Accesses to this register us the fol ing encodings in the System register encoding space:

**MRS <Xt>, IC _PPI_DVIR >_EL2 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0 00 | 0b1100 | 0b1010 | 0b00:n[0] |

```
integer  UInt(op2<0>);

if !(IsFeat Implemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB40 + (8 * n)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_PPI_DVIR_EL2[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_PPI_DVIR_EL2[n];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## MSR ICH_PPI_DVIR<n>_EL2, <Xt> ; Where n = 0-1

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b00:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        NVMem[0xB40 + (8 * n)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_PPI_DVIR_EL2[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_PPI_DVIR_EL2[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.9 ICH_PPI_ENABLER<n>_EL2, Interrupt Controller Virtual Interrupt Enable Registers, n = 0 - 1

The ICH_PPI_ENABLER<n>_EL2 characteristics are:

**Purpose**

Access to Enable state for virtual PPIs.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

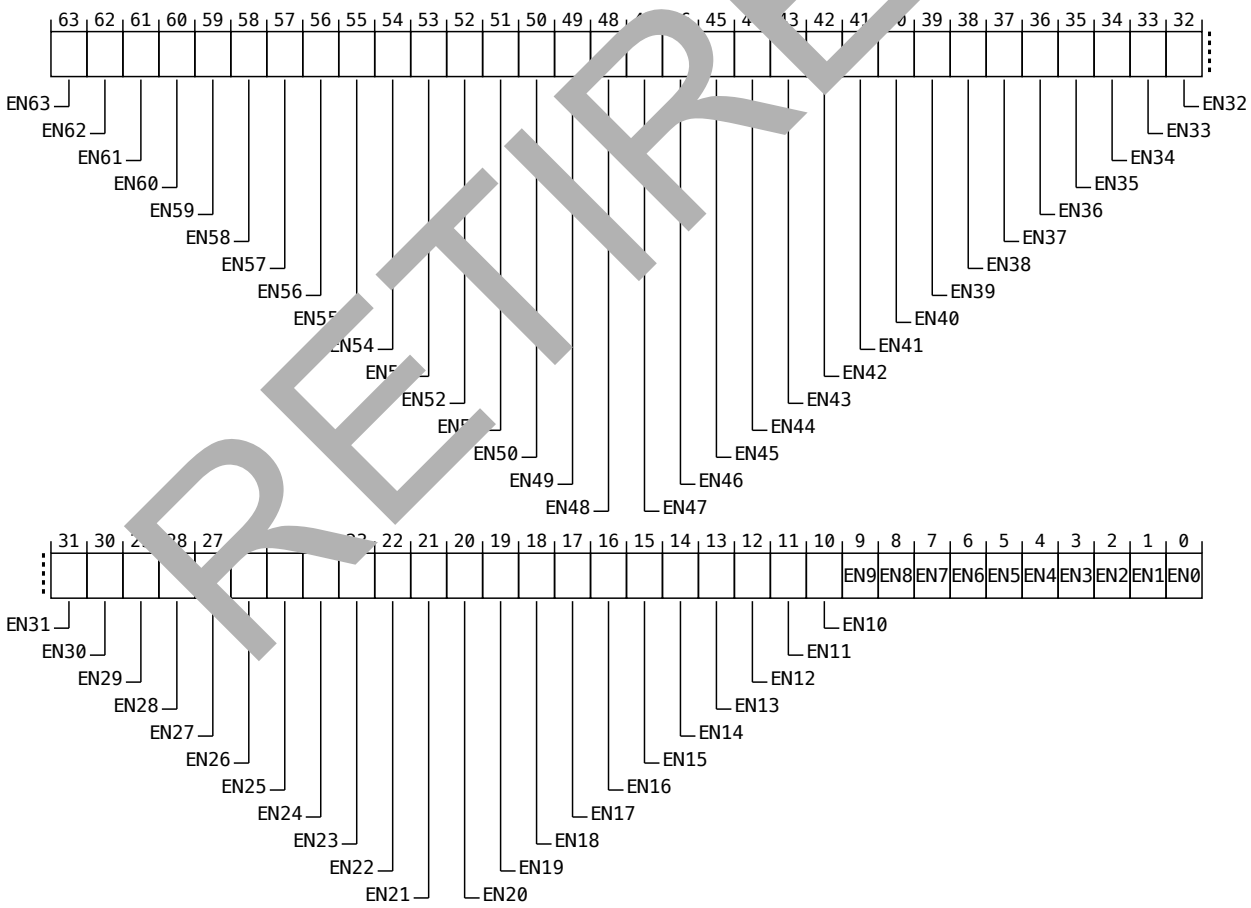
AArch64 System register ICH_PPI_ENABLER<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_ENABLER<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented. [illegible]L2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_PPI_ENABLE[illegible]n>_EL2 a[illegible] UNDEFINED.

**Attributes**

ICH_PPI_ENABLER<n>_EL2 is a 64-bit register.

## Field descriptions




**EN<x> , bits [x], for x = 63 to 0**

Alias of equivalent ICV_PPI_ENABLER<n>_EL1 field.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RW.

### Accessing ICH_PPI_ENABLER<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_PPI_ENABLER<n>_EL2 ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b01:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB50 + (8 * n)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_PPI_ENABLER_EL2[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_PPI_ENABLER_EL2[n];
```

**MSR ICH_PPI_ENABLER<n>_EL2, <Xt> ; Where n = 0-1**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b01:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB50 + (8 * n)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_PPI_ENABLER_EL2[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_PPI_ENABLER_EL2[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.10 ICH_PPI_PENDR<n>_EL2, Interrupt Controller Virtual Interrupt Pending State Registers, n = 0 - 1

The ICH_PPI_PENDR<n>_EL2 characteristics are:

**Purpose**

Pending state for virtual PPIs.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register ICH_PPI_PENDR<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_CPENDR<n>_EL1[63:0].

AArch64 System register ICH_PPI_PENDR<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_SPENDR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_PPI_PENDR<n>_EL2 are UNDEFINED.

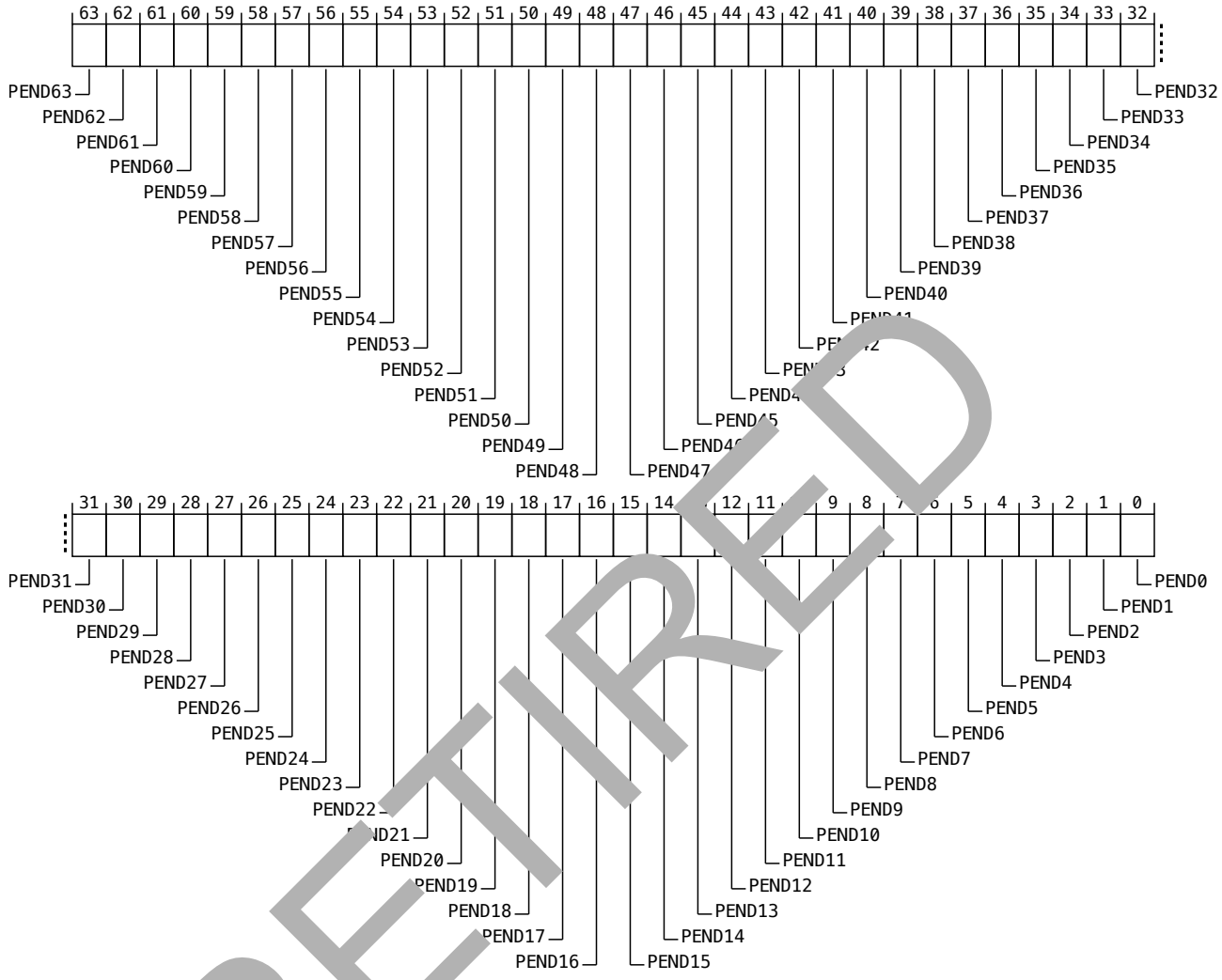
**Attributes**

ICH_PPI_PENDR<n>_EL2 is a 64-bit register.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Field descriptions




**PEND<x> , bit [x], for x = 63 to 0**

PEND<x> accesses the Pending state of virtual PPI <x>.

| PEND<x> | Meaning |
|---|---|
| 0b0 | Not pending |
| 0b1 | Pending |

Accessing this field has the following behavior:

- When !IsPPIImplemented((n * 64) + x), access to this field is RES0.
- Otherwise, access to this field is RW.

## Accessing ICH_PPI_PENDR<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_PPI_PENDR<n>_EL2 ; Where n = 0–1**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_PPI_PENDR_EL2[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_PPI_PENDR_EL2[n];
```

## MSR ICH_PPI_PENDR<n>_EL2, <Xt> ; where n = 0-1

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1010 | 0b10:n[0] |

```
integer n = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_PPI_PENDR_EL2[n] = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_PPI_PENDR_EL2[n] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.11 ICH_PPI_PRIORITYR<n>_EL2, Interrupt Controller Virtual Interrupt Priority Registers, n = 0 - 15

The ICH_PPI_PRIORITYR<n>_EL2 characteristics are:

**Purpose**

Priority of virtual PPIs.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

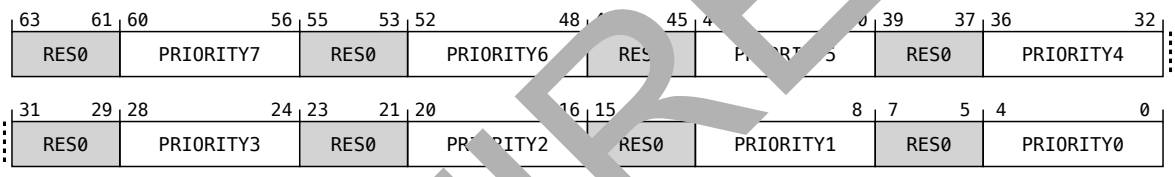
AArch64 System register ICH_PPI_PRIORITYR<n>_EL2 bits [63:0] are architecturally mapped to AArch64 System register ICV_PPI_PRIORITYR<n>_EL1[63:0].

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_PPI_PRIORITYR<n>_EL2 are UNDEFINED.

**Attributes**

ICH_PPI_PRIORITYR<n>_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:61, 55:53, 47:45, 39:37, 31:29, 23:21, 15:13, 7:5]**

Reserved, RES0.

**PRIORITY<x> , bits [60:56, 52:48, 44:40, 36:32, 28:24, 20:16, 12:8, 4:0], for x = 7 to 0**

Alias of equivalent ICV_PPI_PRIORITY<n>_EL1 field.

Fields corresponding to unimplemented INTIDs are RES0.

Accessing this field has the following behavior:

- When IsPPIImplemented((n * 8) + x), access to this field is RES0.
  Otherwise, access to this field is RW.

## Accessing ICH_PPI_PRIORITYR<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_PPI_PRIORITYR<n>_EL2 ; Where n = 0-15**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b111:n[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        X[t, 64] = NVMem[0xB80 + (8 * n)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_PPI_PRIORITYR_EL2[n];
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_PPI_PRIORITYR_EL2[n];
```

## MSR ICH_PPI_PRIORITYR<n>_EL2, <Xt> ; Where n = 0-15

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b11[3] | n[2:0] |

```
integer n = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_GCIE) && (Hav[illegible]) || [illegible]eEL([illegible])) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'[illegible]'} && [illegible]sFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented[illegible]EAT_[illegible]E_LEGA[illegible]) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB80 + (8 * n)[illegible] = X[t, 64[illegible]
    elsif EffectiveHCR_EL2_[illegible]() IN {'xx1'[illegible]&& (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImpl[illegible]ente[illegible]FEAT_GCIE_LE[illegible]CY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap[illegible]2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL [illegible]EL2 the[illegible]
    ICH_PPI_PR[illegible]ITYR_EL[illegible]n] = X[t, 64];
elsif PSTATE.EL [illegible]EL3[illegible]hen
    ICH_PPI_PRIOR[illegible]EL2[n] = [illegible]t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.12 ICH_VCTLR_EL2, Interrupt Controller Virtual CPU interface Control Register

The ICH_VCTLR_EL2 characteristics are:

**Purpose**

Controls behavior of the Virtual CPU interface.

**Configuration**

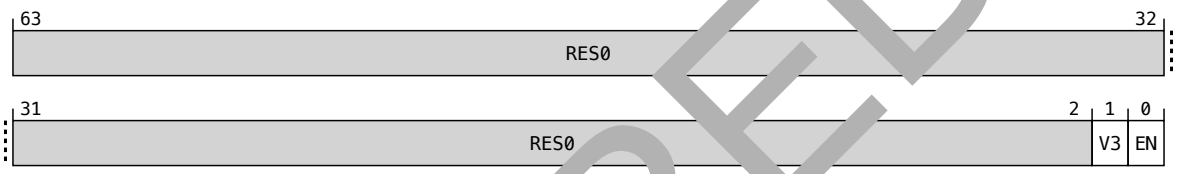
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCIE is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VCTLR_EL2 are UNDEFINED.

**Attributes**

ICH_VCTLR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:2]**

Reserved, RES0.

**V3, bit [1]**

**When FEAT_GCIE_LEGACY is implemented:**

Enable bit for Legacy operation.

If EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}, the Effective value of this field is

When the Effective value of this field is 0, the Effective value of ICH_HCR_EL2.En is 0.

See 'Legacy virtual CPU interface' for more information.

| V3 | Meaning |
|---|---|
| 0b0 | Legacy operation disabled. |
| 0b1 | Legacy operation enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**EN, bit [0]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When FEAT_GCIE_LEGACY is not implemented or (FEAT_GCIE_LEGACY is implemented and ICH_VCTLR_EL2.V3 == 0):**

Enable interrupts for the Virtual Interrupt Domain.

If EL2 is not enabled in the Security state identified by SCR_EL3.{NSE,NS}, the Effective value of this field is 0.

| EN | Meaning |
|---|---|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

When this field is 0, there is no HPPI of Sufficient priority f the Virtual terrupt Domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKN VN val .

**Otherwise:**

Reserved, RES0.

## Accessing ICH_VCTLR_EL2

Accesses to this register use the following e odings i System register encoding space:

**MRS <Xt>, ICH_VCTLR_EL**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 1100 | 0b1011 | 0b100 |

```
if !(IsFeatureI lement (FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PST EL == then
    UND INED;
elsi STATE.E == EL1
    f Effect eHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪ sFeatu plemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        , 64] = NVMem[0xB28];
    elsi ffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪ FeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch .SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ICH_VCTLR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICH_VCTLR_EL2;
```

**MSR ICH_VCTLR_EL2, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b100 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !(IsFeatureImplemented(FEAT_GCIE) && (HaveEL(EL2) || HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        NVMem[0xB28] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (!IsFeatureImplemented(FEAT_GCIE_LEGACY) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '0')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICH_VCTLR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ICH_VCTLR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.6.13 ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR_EL2 characteristics are:

**Purpose**

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

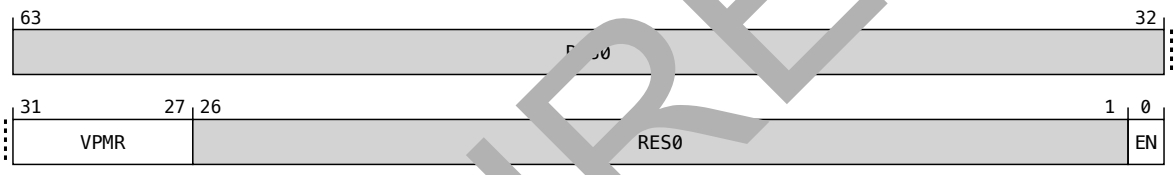
This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_VMCR_EL2 are UNDEFINED.

**Attributes**

ICH_VMCR_EL2 is a 64-bit register.

## Field descriptions

**When IsFeatureImplemented(FEAT_GCIE) && ((!IsFeatureImplemented(FEAT_GCIE_LEGACY)) || (IsFeatureImplemented(FEAT_GCIE_LEGACY) && (ICH_VCTLR_EL2.V3 == '0'))):**




**Bits [63:32]**

Reserved, RES0.

**VPMR, bits [31:27]**

Virtual priority mask.

This field is an alias of ICV_PMR_EL1.Priority.

**Bits [26:1]**

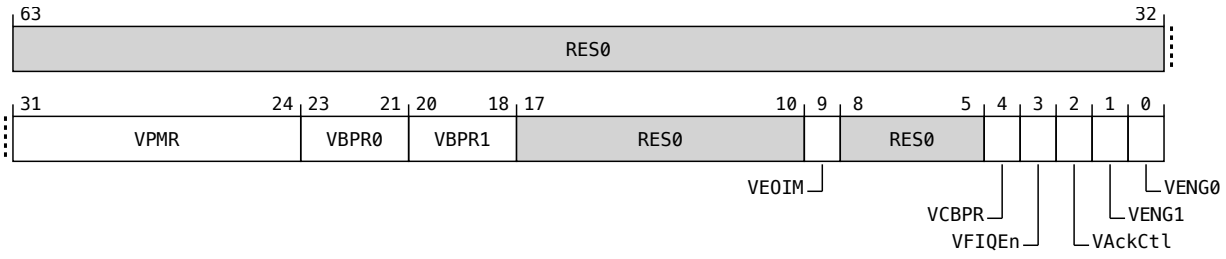
Reserved, RES0.

**EN, bit [0]**

Enable interrupts for the Virtual Interrupt Domain. When this field is 0, there is no HPPI of Sufficient priority for the Virtual Interrupt Domain.

This field is an alias of ICV_CR0_EL1.EN.

**Otherwise:**




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [63:32]**

Reserved, RES0.

**VPMR, bits [31:24]**

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of ICV_PMR_EL1.Priority.

**VBPR0, bits [23:21]**

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 1.

This field is an alias of ICV_BPR0_EL1.BinaryPoint.

The minimum value of this field is determined by ICH_VTR_EL2.PREbits. An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

**VBPR1, bits [20:18]**

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 0.

This field is an alias of ICV_BPR1_EL1.BinaryPoint.

This field is always accessible to EL2 accesses regardless of the setting of the ICH_VMCR_EL2.VCBPR field.

For Secure writes, the minimum value of this field is the minimum value of ICH_VMCR_EL2.VBPR0.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

**Bits [17:10]**

Reserved, RES0.

**VEOIM, bit [9]**

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt.

| VEOIM | Meaning |
|---|---|
| 0b0 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality. |

This bit is an alias of ICV_CTLR_EL1.EOImode.

**Bits [8:5]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**VCBPR, bit [4]**

Virtual Common Binary Point Register. Possible values of this bit are:

| VCBPR | Meaning |
|---|---|
| 0b0 | ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts. |
| 0b1 | Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Writes to ICV_BPR1_EL1 are ignored. |

This field is an alias of ICV_CTLR_EL1.CBPR.

**VFIQEn, bit [3]**

Virtual FIQ enable. Possible values of this bit are:

| VFIQEn | Meaning |
|---|---|
| 0b0 | Group 0 virtual interrupts are presented as virtual IRQs. |
| 0b1 | Group 0 virtual interrupts are presented as virtual FIQs. |

This bit is an alias of GICV_CTLR.FIQ.

In implementations where the Non-secure copy of ICC_SRE_EL1.SRE is always 1, this bit is RES1.

**VAckCtl, bit [2]**

Virtual AckCtl. Possible values of this bit are:

| VAckCtl | Meaning |
|---|---|
| 0b0 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022. |
| 0b1 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt. |

This bit is an alias of GICV_CTLR.AckCtl.

This field is supported for backward compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of ICC_SRE_EL1.SRE is always 1, this bit is RES0.

**VENG1, bit [1]**

Virtual Group 1 interrupt enable. Possible values of this bit are:

| VENG1 | Meaning |
|---|---|
| 0b0 | Virtual Group 1 interrupts are disabled. |
| 0b1 | Virtual Group 1 interrupts are enabled. |

This bit is an alias of ICV_IGRPEN1_EL1.Enable.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**VENG0, bit [0]**

Virtual Group 0 interrupt enable. Possible values of this bit are:

| VENG0 | Meaning |
|---|---|
| 0b0 | Virtual Group 0 interrupts are disabled. |
| 0b1 | Virtual Group 0 interrupts are enabled. |

This bit is an alias of ICV_IGRPEN0_EL1.Enable.

## Accessing ICH_VMCR_EL2

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_VMCR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE)) && (HaveEL(EL2) ||
    ↪HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'11'} then
        X[t, 64] = NVMem[0x4C8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_VMCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_VMCR_EL2;
```

**MSR ICH_VMCR_EL2, <Xt>**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE)) && (HaveEL(EL2) ||
    ↪HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x4C8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_VMCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_VMCR_EL2 = X[t, 64];
```

### 9.6.14 Nested virtualization

$I_{XDKPJ}$ The Arm architecture's enhanced support for nested virtualization (FEAT_NV2) provides a mechanism for hardware to transform reads and writes from System registers into loads and writes from memory.

See "Enhanced support for nested virtualization" in the Architecture Reference Manual for A-profile [1].

$R_{KZTGX}$ When System register accesses are transformed to memory accesses, the D8-63 (Memory address offset associated with transformed register access) in the Architecture Reference Manual for A-profile [1] describes the offsets used for each affected regiser. This table is extended as follows to cover the GICv5 System registers:

| Register access if HCR_EL2.NV1 is 0 | Register access if HCR_EL2.NV1 is 1 | Memory offset |
|---|---|---|
| ICH_APR_EL2 | ICH_APR_EL2 | 0xB00 |
| ICH_CONTEXTR_EL2 | ICH_CONTEXTR_EL2 | 0xB08 |
| ICH_HFGITR_EL2 | ICH_HFGITR_EL2 | 0xB10 |
| ICH_HFGRTR_EL2 | ICH_HFGRTR_EL2 | 0xB18 |
| ICH_HFGWTR_EL2 | ICH_HFGWTR_EL2 | 0xB20 |
| ICH_PPI_ACTIVER<n>_EL2 | ICH_PPI_ACTIVER<n>_EL2 | 0xB30 + 8*n |
| ICH_PPI_DVIR<n>_EL2 | ICH_PPI_DVIR<n>_EL2 | 0xB40 + 8*n |
| ICH_PPI_ENABLER<n>_EL2 | ICH_PPI_ENABLER<n>_EL2 | 0xB50 + 8*n |
| ICH_PPI_PRIORITYR<n>_EL2 | ICH_PPI_PRIORITYR<n>_EL2 | 0xB80 + 8*n |
| ICH_VCTLR_EL2 | ICH_VCTLR_EL2 | 0xB28 |
| ICH_VMCR_EL2 | ICH_VMCR_EL2 | 0x4C8 |

Offsets for registers introduced by GICv3 or GICv4 are unchanged.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 9.7 Legacy hypervisor control registers

Registers to manage operation of the Legacy virtual CPU interface.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 9.7.1 ICH_AP0R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n>_EL2 characteristics are:

**Purpose**

Provides information about Group 0 virtual active priorities for EL2.

**Configuration**

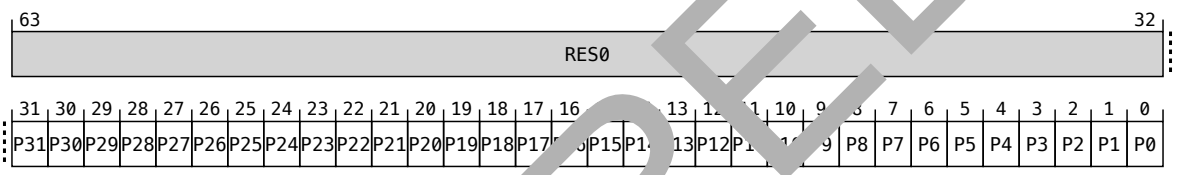
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_AP0R<n>_EL2 are UNDEFINED.

**Attributes**

ICH_AP0R<n>_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**P<x> , bits [x], for x = 31 to**

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

| P<x> | Meaning |
|---|---|
| 0b0 | There is no Group 0 interrupt active with this priority level, or all active Group 0 interrupts with this priority level have undergone priority-drop. |
| 0b1 | There is a Group 0 interrupt active with this priority level which has not undergone priority drop. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0_EL2 in the bits corresponding to 00:Priority[5:1].

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The active state of preemption levels 64 - 126 are held in ICH_AP0R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3_EL2 in the bits corresponding to 11:Priority[5:1].

---

**Note**

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

---

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Additional Information**

If FEAT_GICv3 is implemented, software must ensure that ICH_AP0R<n>_EL2 is 0 for legacy VMs otherwise behavior is UNPREDICTABLE. For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® GIC Architecture Specification, GIC architecture version 3.0 and version 4.1 (ARM IHI 0069).

The active priorities for Group 0 and Group 1 interrupts for legacy VMs are held in ICH_AP1R<n>_EL2 and reads and writes to GICV_AP1R<n>_EL1 access ICH_AP1R<n>_EL2. This means that ICH_AP0R<n>_EL2 is inaccessible to legacy VMs.

## Accessing ICH_AP0R<n>_EL2

If FEAT_GCIE_LEGACY is implemented, only 32 priority levels are supported, which means that ICH_AP0R0_EL2 is the only implemented register.

ICH_AP0R1_EL2 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

---

**Note**

The number of preemption is indicated by ICH_VTR_EL2.PREbits

---

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- ICH_AP1R<n>_EL2.

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

Accesses to this register use the following encodings in the System register encoding space:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### MRS <Xt>, ICH_AP0R<m>_EL2 ; Where m = 0-3

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b0:m[1:0] |

```
integer m = UInt(op2<1:0>);

if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        X[t, 64] = NVMem[0x480 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_AP0R_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_AP0R_EL2[m];
```

### MSR ICH_AP0R<m>_EL2, <Xt> ; Where m = 0-3

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b0:m[1:0] |

```
integer m = UInt(op2<1:0>);

if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        NVMem[0x480 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP0R_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP0R_EL2[m] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.2 ICH_AP1R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n>_EL2 characteristics are:

**Purpose**

Provides information about Group 1 virtual active priorities for EL2.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

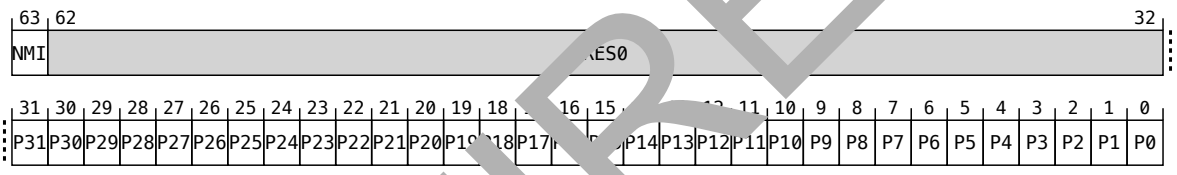
When FEAT_GCIE_LEGACY is implemented, AArch64 System register ICH_AP1R0_EL2 bits [31:0] are architecturally mapped to AArch64 System register ICH_APR_EL2[31:0].

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_AP1R<n>_EL2 are UNDEFINED.

**Attributes**

ICH_AP1R<n>_EL2 is a 64-bit register.

## Field descriptions




**NMI, bit [63]**

**When (FEAT_GICv3_NMI implemented or FEAT_GCIE_LEGACY is implemented) and n == 0:**

Indicates whether the running virtual priority is from a NMI.

| NMI | Meaning |
|---|---|
| 0b0 | There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop. |
| 0b1 | There is an active Group 1 NMI. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Reserved, RES0.

**Bits [62:32]**

Reserved, RES0.

**P<x> , bits [x], for x = 31 to 0**

Group 1 interrupt active priorities. Possible values of each bit are:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| P<x> | Meaning |
|---|---|
| 0b0 | There is no Group 1 interrupt active with this priority level, or all active Group 1 interrupts with this priority level have undergone priority-drop. |
| 0b1 | There is a Group 1 interrupt active with this priority level which has not undergone priority drop. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP1R3_EL2 in the bits corresponding to 11:Priority[5:1].

**Note**

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

On a Warm reset, this field resets to '0'.

**Additional Information**

If FEAT_GICv3 is implemented, this register is always used for legacy VMs, regardless of the group of the virtual interrupt. Reads and writes to GICV_APR<n> access ICH_AP1R<n>_EL2. For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

## Accessing ICH_AP1R<n>_EL2

If FEAT_GCIE_LEGACY is implemented, only 32 priority levels are supported, which means that ICH_AP1R0_EL2 is the only implemented register.

ICH_AP1R1_EL2 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP1R2_EL2 and ICH_AP1R3_EL2 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Note**

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- ICH_AP1R<n>_EL2.

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_AP1R<m>_EL2 ; Where m = 0-3**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b0:m[1:0] |

```
integer m = UInt(op2<1:0>);

if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ExecutionEHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        X[t, 64] = NVMem[0x4A0 + (8 * m)];
    elsif ExecutionEL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        (IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_AP1R_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_AP1R_EL2[m];
```

**MSR ICH_AP1R<m>_EL2, <Xt> ; Where m = 0-3**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b0:m[1:0] |

```
integer m = UInt(op2<1:0>);

if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        NVMem[0x4A0 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP1R_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP1R_EL2[m] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.3 ICH_EISR_EL2, Interrupt Controller End of Interrupt Status Register

The ICH_EISR_EL2 characteristics are:

**Purpose**

Indicates which List registers have outstanding EOI maintenance interrupts.

**Configuration**

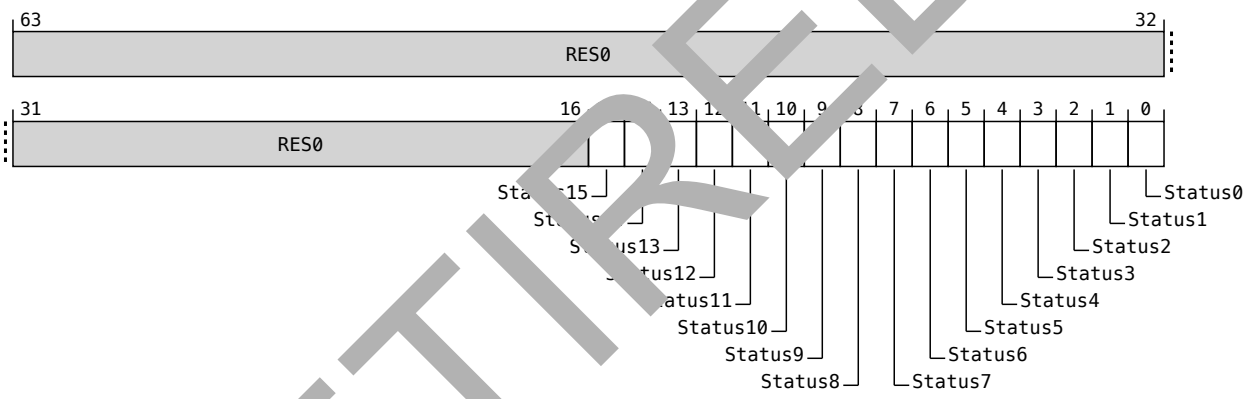
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_EISR_EL2 are UNDEFINED.

**Attributes**

ICH_EISR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 15 to 0**

EOI maintenance interrupt status bit for List register <n>:

| Status<n> | Meaning |
|---|---|
| 0b0 | List register <n>, ICH_LR<n>_EL2, does not have an EOI maintenance interrupt. |
| 0b1 | List register <n>, ICH_LR<n>_EL2, has an EOI maintenance interrupt that has not been handled. |

For any ICH_LR<n>_EL2, the corresponding status bit is set to 1 if all of the following are true:

- ICH_LR<n>_EL2.State is 0b00.
- ICH_LR<n>_EL2.HW is 0.
- ICH_LR<n>_EL2.EOI (bit [41]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

Otherwise the status bit takes the value 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Accessing ICH_EISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_EISR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b011 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_EISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_EISR_EL2;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.4 ICH_ELRSR_EL2, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR_EL2 characteristics are:

**Purpose**

These registers can be used to locate a usable List register when the hypervisor is delivering an interrupt to a VM.

**Configuration**

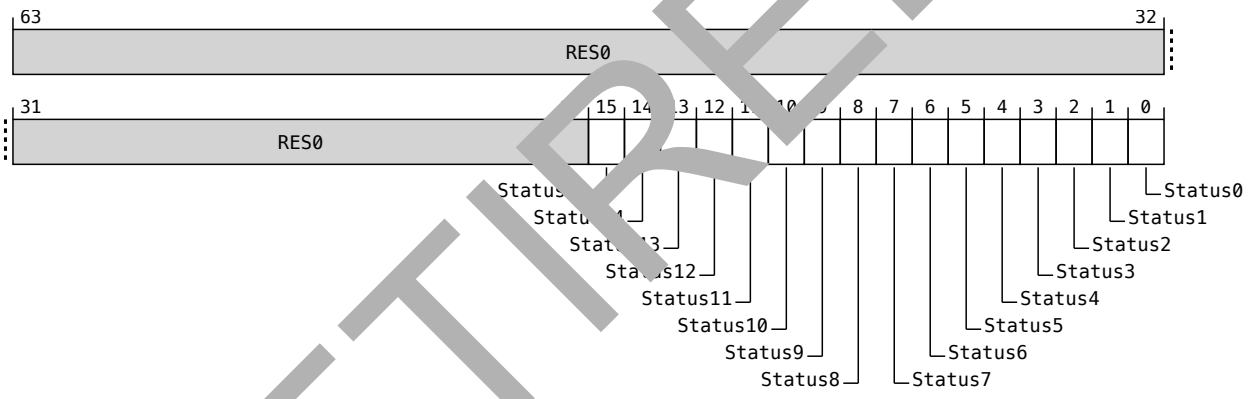
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_ELRSR_EL2 are UNDEFINED.

**Attributes**

ICH_ELRSR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 15 to 0**

Status bit for List register <n>, ICH_LR<n>_EL2:

| Status<n> | Meaning |
|---|---|
| 0b0 | List register ICH_LR<n>_EL2, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt. |
| 0b1 | List register ICH_LR<n>_EL2 does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt. |

For any List register <n>, the corresponding status bit is set to 1 if ICH_LR<n>_EL2.State is 0b00 and either ICH_LR<n>_EL2.HW is 1 or ICH_LR<n>_EL2.EOI (bit [41]) is 0.

Otherwise the status bit takes the value 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Accessing ICH_ELRSR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_ELRSR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b101 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_ELRSR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_ELRSR_EL2;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.5 ICH_HCR_EL2, Interrupt Controller Hyp Control Register

The ICH_HCR_EL2 characteristics are:

**Purpose**

Controls the environment for the virtual CPU interface.

**Configuration**

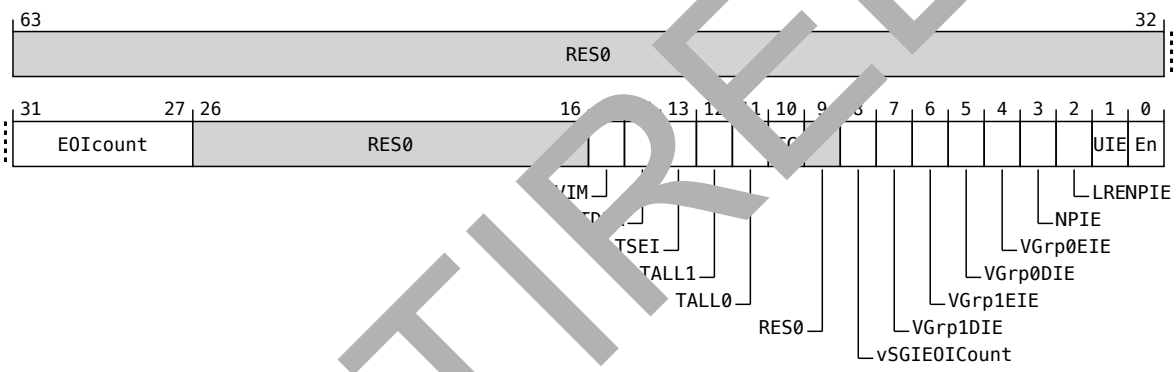
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_HCR_EL2 are UNDEFINED.

**Attributes**

ICH_HCR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**EOIcount, bits [31:27]**

This field increments whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when VEOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when VEOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers (ICH_AP0R<n>_EL2/ICH_AP1R<n>_EL2) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000'.

**Bits [26:16]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**DVIM, bit [15]**

**When GICv3 is implemented and ICH_VTR_EL2.DVIM == 1:**

Directly-injected Virtual Interrupt Mask.

| DVIM | Meaning |
|---|---|
| 0b0 | This control has no effect on the signaling of virtual interrupts. |
| 0b1 | Virtual interrupts received via direct-injection are not presented to the virtual CPU interface and not considered when determining the highest priority pending virtual interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Reserved, RES0.

**TDIR, bit [14]**

**When FEAT_GICv3_TDIR is implemented:**

Trap EL1 writes to ICC_DIR_EL1 and ICV_DIR_EL1.

| TDIR | Meaning |
|---|---|
| 0b0 | EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are not trapped to EL2, unless trapped by other mechanisms. |
| 0b1 | EL1 writes of ICC_DIR_EL1 are trapped to EL2. It is IMPLEMENTATION DEFINED whether writes of ICC_DIR_EL1 are trapped. Not trapping ICC_DIR_EL1 writes is DEPRECATED. |

Arm deprecates not including this trap bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Reserved, RES0.

**TSEI, bit [13]**

**When GICv3 is implemented:**

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at EL1.

| TSEI | Meaning |
|---|---|
| 0b0 | Locally generated SEIs do not cause a trap to EL2. |
| 0b1 | Locally generated SEIs trap to EL2. |

If ICH_VTR_EL2.SEIS is 0, this bit is RES0.

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Reserved, RES0.

**TALL1, bit [12]**

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

| TALL1 | Meaning |
|---|---|
| 0b0 | EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal. |
| 0b1 | EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**TALL0, bit [11]**

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

| TALL0 | Meaning |
|---|---|
| 0b0 | EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal. |
| 0b1 | EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**TC, bit [10]**

Trap all EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

| TC | Meaning |
|---|---|
| 0b0 | EL1 accesses to common registers proceed as normal. |
| 0b1 | EL1 accesses to common registers trap to EL2. |

This affects accesses to ICC_SGI0R_EL1, ICC_SGI1R_EL1, ICC_ASGI1R_EL1, ICC_CTLR_EL1, ICC_DIR_EL1, ICC_PMR_EL1, ICC_RPR_EL1, ICV_CTLR_EL1, ICV_DIR_EL1, ICV_PMR_EL1, and ICV_RPR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Bit [9]**

Reserved, RES0.

**vSGIEOICount, bit [8]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When GICv4.1 is implemented:**

Controls whether deactivation of virtual SGIs can increment ICH_HCR_EL2.EOIcount

| vSGIEOICount | Meaning |
|---|---|
| 0b0 | Deactivation of virtual SGIs can increment ICH_HCR_EL2.EOIcount. |
| 0b1 | Deactivation of virtual SGIs does not increment ICH_HCR_EL2.EOIcount. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Otherwise:**

Reserved, RES0.

**VGrp1DIE, bit [7]**

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp1DIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp1EIE, bit [6]**

VM Group 1 Enable Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp1EIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp0DIE, bit [5]**

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp0DIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| VGrp0DIE | Meaning |
|---|---|
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp0EIE, bit [4]**

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp0EIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**NPIE, bit [3]**

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

| NPIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled while the List registers contain no interrupts in the pending state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**LRENPIE, bit [2]**

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

| LRENPIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted while the EOIcount field is not 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**UIE, bit [1]**

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

| UIE | Meaning |
|---|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**En, bit [0]**

Enable. Global enable bit for the virtual CPU interface:

| En | Meaning |
|---|---|
| 0b0 | Virtual CPU interface operation disabled. |
| 0b1 | Virtual CPU interface operation enabled. |

If FEAT_GCIE_LEGACY is implemented and ICH_VCTLR_EL2.V3 is 0, the Effective value of this field is 0.

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of ICV_IAR0_EL1, ICV_IAR1_EL1, ICV_NMIAR1_EL1, GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

**Note**

This field is RES0 when SCR_EL3.{NS,EEL2}=={0,0}

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

## Accessing ICH_HCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_HCR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b000 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        X[t, 64] = NVMem[0x4C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_HCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_HCR_EL2;
```

## MSR ICH_HCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b000 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        NVMem[0x4C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_HCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_HCR_EL2 = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.6 ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n>_EL2 characteristics are:

**Purpose**

Provides interrupt context information for the virtual CPU interface.

**Configuration**

If EL2 is not implemented, this register is RES0 from EL3.

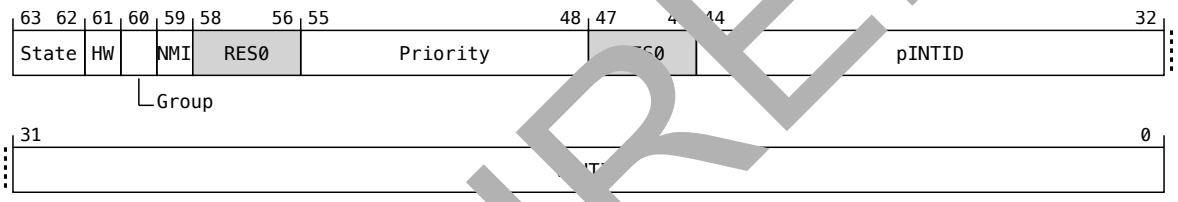
If list register n is not implemented, then accesses to this register are UNDEFINED.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_LR<n>_EL2 are UNDEFINED.

**Attributes**

ICH_LR<n>_EL2 is a 64-bit register.

## Field descriptions




**State, bits [63:62]**

The state of the interrupt:

| State | Meaning |
|---|---|
| 0b00 | Invalid (Inactive). |
| 0b01 | Pending. |
| 0b10 | Active. |
| 0b11 | Pending and active. |

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. When a virtual interrupt from a List register is acknowledged, the state is updated to the active state regardless of the interrupt type.

Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**HW, bit [61]**

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the ID that the pINTID field indicates.

| HW | Meaning |
|---|---|
| 0b0 | The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated. |
| 0b1 | If FEAT_GICv3 is implemented, all of the following are true:<br>• The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical interrupt ID.<br>• If ICH_VMCR_EL2.VEOIM is 0, this request corresponds to a write to ICC_EOIR0_EL1 or ICC_EOIR1_EL1. Otherwise, it corresponds to a write to ICC_DIR_EL1.<br>If FEAT_GCIE_LEGACY is implemented, the virtual interrupt also deactivates the physical interrupt specified in pINTID. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Group, bit [60]**

Indicates the group for this virtual interrupt.

| Group | Meaning |
|---|---|
| 0b0 | This is a Group 0 virtual interrupt. ICH_VMCR_EL2.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR_EL2.VENG0 enables signaling of this interrupt to the virtual machine. |
| 0b1 | This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR_EL2.VENG1 enables the signaling of this interrupt to the virtual machine.<br>If ICH_VMCR_EL2.VCBPR is 0, then ICC_BPR1_EL1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n>_EL2 determines preemption. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NMI, bit [59]**

**When FEAT_GICv3_NMI is implemented:**

Indicates whether the virtual priority has the non-maskable property.

| NMI | Meaning |
|---|---|
| 0b0 | vINTID does not have the non-maskable interrupt property. |
| 0b1 | vINTID has the non-maskable interrupt property. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Setting ICH_LR<n>_EL2.NMI to 1 when ICH_LR<n>_EL2.State is not Invalid is CONSTRAINTED UNPREDICTABLE if either ICH_LR<n>_EL2.vINTID indicates an LPI or ICH_LR<n>_EL2.Group is 0.

The permitted behaviors are:

- ICH_LR<n>_EL2.NMI is treated as 0 for all purposes other than a direct read of the register.
- The virtual interrupt is presented with superpriority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [58:56]**

Reserved, RES0.

**Priority, bits [55:48]**

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[48] up to bit[50]. The number of implemented bits can be discovered from ICH_VTR_EL2 bits.

When ICH_LR<n>_EL2.NMI is set to 1, this field is RES0 and the virtual interrupt's priority is treated as 0x00.

If FEAT_GCIE_LEGACY is implemented, 5 bits of priority are implemented meaning that bits[50:48] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bits [47:45]**

Reserved, RES0.

**pINTID, bits [44:32]**

**When FEAT_GCIE_LEGACY is implemented:**

Physical PPI INTID, for hardware interrupts.

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42]: RES0.
- Bit[41]: EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
- Bits[40:32]: RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

- This field specifies the ID of a physical PPI in the Current Physical Interrupt Domain.
- This field is only required to implement enough bits to hold a valid value for the implemented physical PPIs. Any unused higher order bits are RES0.
- If pINTID specifies an unreachable PPI, no physical interrupt is deactivated.

**Otherwise:**

Physical INTID, for hardware interrupts.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42]: RES0.
- Bit[41]: EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
- Bits[40:32]: RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
- When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with the same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR_EL1.IDbits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

**vINTID, bits [31:0]**

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LR<n>_EL2.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LR<n>_EL2.State == 0b01.
- ICH_LR<n>_EL2.State == 0b10.
- ICH_LR<n>_EL2.State == 0b11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR_EL2.IDbits.

When ICC_SRE_EL1.SRE == 0, specifying a vINTID in the LPI range is UNPREDICTABLE

---

**Note**

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

---

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

## Accessing ICH_LR<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_LR<m>_EL2 ; Where m = 0-15
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b110:m[3] | m[2:0] |

```
integer m = UInt(CRm<0>:op2<2:0>);

if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemente  EAT_GI  3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTL  L2.V3 == '1')) then
        X[t, 64] = NVMem[0x400 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeature  pleme  ed(FEAT_GI  3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && IC  VCTLR_EL  3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_LR_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3,  18);
    else
        X[t, 64] = ICH_LR_EL  [m];
```

## MSR ICH_LR<m>_EL2, <Xt> ; Where m = 0-15

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b110:m[3] | m[2:0] |

```
int  r m =  Rm<0>:op2<2:0>);

if !((Is  atureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| veEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        NVMem[0x400 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_LR_EL2[m] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_LR_EL2[m] = X[t, 64];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.7 ICH_MISR_EL2, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR_EL2 characteristics are:

**Purpose**

Indicates which maintenance interrupts are asserted.

**Configuration**

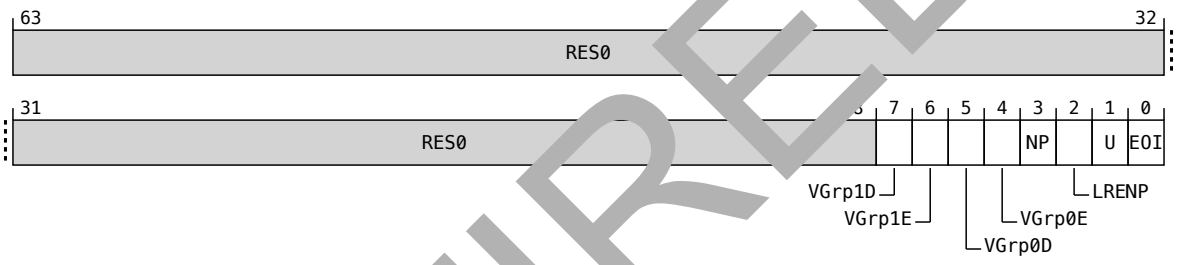
If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_MISR_EL2 are UNDEFINED.

**Attributes**

ICH_MISR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:8]**

Reserved, RES0.

**VGrp1D, bit [7]**

vPE Group 1 Disabled

| VGrp1D | Meaning |
|---|---|
| 0b0 | vPE Group 1 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp1DIE==1 and ICH_VMCR_EL2.VENG1==is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp1E, bit [6]**

vPE Group 1 Enabled.

| VGrp1E | Meaning |
|---|---|
| 0b0 | vPE Group 1 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Enabled maintenance interrupt asserted. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp1EIE==1 and ICH_VMCR_EL2.VENG1==is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp0D, bit [5]**

vPE Group 0 Disabled.

| VGrp0D | Meaning |
|---|---|
| 0b0 | vPE Group 0 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp0DIE==1 and ICH_VMCR_EL2.VENG0==0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**VGrp0E, bit [4]**

vPE Group 0 Enabled.

| VGrp0E | Meaning |
|---|---|
| 0b0 | vPE Group 0 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp0EIE==1 and ICH_VMCR_EL2.VENG0==1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**NP, bit [3]**

No Pending.

| NP | Meaning |
|---|---|
| 0b0 | No Pending maintenance interrupt not asserted. |
| 0b1 | No Pending maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.NPIE==1 and no List register is in pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**LRENP, bit [2]**

List Register Entry Not Present.

| LRENP | Meaning |
|---|---|
| 0b0 | List Register Entry Not Present maintenance interrupt not asserted. |
| 0b1 | List Register Entry Not Present maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.LRENPIE==1 and ICH_HCR_EL2.EOIcount is nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**U, bit [1]**

Underflow.

| U | Meaning |
|---|---|
| 0b0 | Underflow maintenance interrupt not asserted. |
| 0b1 | Underflow maintenance interrupt asserted. |

This maintenance interrupt is asserted when ICH_HCR_EL2.UIE==1 and zero or one of the List register entries are marked as a valid interrupt, that is if the corresponding ICH_LR<n>_EL2.State bits do not equal 0x0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**EOI, bit [0]**

End Of Interrupt.

| EOI | Meaning |
|---|---|
| 0b0 | End Of Interrupt maintenance interrupt not asserted. |
| 0b1 | End Of Interrupt maintenance interrupt asserted. |

This maintenance interrupt is asserted when at least one bit in ICH_EISR_EL2 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

**Additional Information**

The U and NP bits do not include the status of any pending/active 'VSet (IRI)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069) packets because these bits control generation of interrupts that allow software management of the contents of the List Registers (which are not affected by 'VSet (IRI)' packets).

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Accessing ICH_MISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_MISR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b010 |

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_MISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_MISR_EL2;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 9.7.8 ICH_VTR_EL2, Interrupt Controller VGIC Type Register

The ICH_VTR_EL2 characteristics are:

**Purpose**

Reports supported GIC virtualization features.

**Configuration**

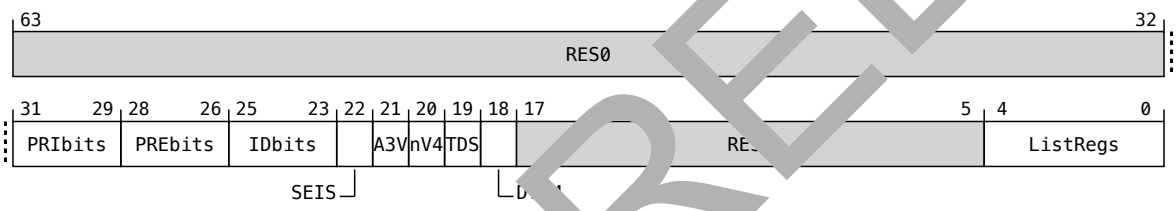
If EL2 is not implemented, this register is RES0 from EL3, except for nV4, which is RES1 from EL3.

This register is present only when (IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2) || HaveEL(EL3)). Otherwise, direct accesses to ICH_VTR_EL2 are UNDEFINED.

**Attributes**

ICH_VTR_EL2 is a 64-bit register.

## Field descriptions




**Bits [63:32]**

Reserved, RES0.

**PRIbits, bits [31:29]**

Priority bits. Indicates the number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is a copy of ICV_CTLR_EL1.PRIbits.

If FEAT_GCIE_LEGACY is implemented, this field returns 4 (5 bits of priority).

The value of this field is an IMPLEMENTATION DEFINED choice of:

| PRIbits | Meaning |
|---|---|
| 0b100..0b110 | The number of virtual priority bits implemented, minus one. |

Access to this field is RO.

**PREbits, bits [28:26]**

Preemption bits. Indicates the number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR_EL2.PRIbits.

This field determines the minimum value of ICH_VMCR_EL2.VBPR0.

If FEAT_GCIE_LEGACY is implemented, this field returns 4 (5 bits of preemption).

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

| PREbits | Meaning |
|---|---|
| 0b000..0b110 | The number of virtual preemption bits implemented, minus one. |

Access to this field is RO.

**IDbits, bits [25:23]**

The number of virtual interrupt identifier bits supported:

The value of this field is an IMPLEMENTATION DEFINED choice of:

| IDbits | Meaning |
|---|---|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

This field is an alias of ICV_CTLR_EL1.IDbits

Access to this field is RO.

**SEIS, bit [22]**

**When FEAT_GCIE_LEGACY is implemented:**

SEI Support. The virtual CPU interface does not support generation of SEIs.

Reads as 0b0

Access to this field is RO.

**Otherwise:**

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

| SEIS | Meaning |
|---|---|
| 0b0 | The virtual CPU interface logic does not support generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports generation of SEIs. |

This bit is an alias of ICV_CTLR_EL1.SEIS.

Access to this field is RO.

**A3V, bit [21]**

Affinity 3 Valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| A3V | Meaning |
|---|---|
| 0b0 | The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The virtual CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers. |

This bit is an alias of ICV_CTLR_EL1.A3V.

Access to this field is RO.

**nV4, bit [20]**

**When FEAT_GCIE_LEGACY is implemented:**

Direct injection of virtual interrupts not supported.

Reads as 0b1

Access to this field is RO.

**Otherwise:**

Direct injection of virtual interrupts not supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| nV4 | Meaning |
|---|---|
| 0b0 | The CPU interface logic supports direct injection of virtual interrupts. |
| 0b1 | The CPU interface logic does not support direct injection of virtual interrupts. |

In GIC, the only permitted value is 0b1.

Access to this field is RO.

**TDS, bit [19]**

**When FEAT_GICv3_TDIR is implemented:**

Separate trapping of EL1 writes to ICV_DIR_EL1 supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| TDS | Meaning |
|---|---|
| 0b0 | Implementation does not support ICH_HCR_EL2.TDIR. |
| 0b1 | Implementation supports ICH_HCR_EL2.TDIR. |

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

Access to this field is RO.

**When FEAT_GCIE_LEGACY is implemented:**

Separate trapping of EL1 writes to ICV_DIR_EL1 supported.

Reads as 0b1

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**DVIM, bit [18]**

**When FEAT_GCIE_LEGACY is implemented:**

Masking of directly-injected virtual interrupts.

Reads as 0b0

Access to this field is RO.

**Otherwise:**

Masking of directly-injected virtual interrupts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| DVIM | Meaning |
|---|---|
| 0b0 | Masking of Directly-injected Virtual Interrupts not supported. |
| 0b1 | Masking of Directly-injected Virtual Interrupts is supported. |

Access to this field is RO.

**Bits [17:5]**

Reserved, RES0.

**ListRegs, bits [4:0]**

List Registers. Indicates the number of List registers implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| ListRegs | Meaning |
|---|---|
| 0b0000..0b1111 | The number of List registers implemented, minus one. |

Access to this field is RO.

## Accessing ICH_VTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

**MRS <Xt>, ICH_VTR_EL2**

| op0 | op1 | CRn | CRm | op2 |
|---|---|---|---|---|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b001 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
if !((IsFeatureImplemented(FEAT_GICv3) || IsFeatureImplemented(FEAT_GCIE_LEGACY)) && (HaveEL(EL2)
    ↪|| HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} && (IsFeatureImplemented(FEAT_GICv3) ||
        ↪(IsFeatureImplemented(FEAT_GCIE_LEGACY) && ICH_VCTLR_EL2.V3 == '1')) then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_VTR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_VTR_EL2;
```

### 9.7.9 Nested virtualization

$I_{HPNKC}$ The Arm architecture's enhanced support for nested virtualization (FEAT_NV2) provides a mechanism for hardware to transform reads and writes from System registers into reads and writes from memory.

When accesses to System registers introduced by GICv3 are transformed into memory accesses, the offsets are unchanged by GICv5. See "Enhanced support for nested virtualization" in the Architecture Reference Manual for A-profile [1].

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# 9.8 Legacy virtual CPU interface registers

Configuration and state of Legacy virtual CPU interface accessible from a GICv3.3 VM.

## 9.8.1 AArch64 Legacy virtual CPU interface registers

$R_{WSWHT}$ When FEAT_GICv3 or FEAT_GCIE_LEGACY is implemented, the following registers are present:

- ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Registers
- ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Registers
- ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0
- ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1
- ICV_CTLR_EL1, Interrupt Controller Virtual Control Register
- ICV_DIR_EL1, Interrupt Controller Deactivate Virtual Interrupt Reg
- ICV_EOIR0_EL1, Interrupt Controller Virtual End Of Interrupt R gister 0
- ICV_EOIR1_EL1, Interrupt Controller Virtual End Of Interrup egister 1
- ICV_HPPIR0_EL1, Interrupt Controller Virtual Highest Pri ority nding Interr t Register 0
- ICV_HPPIR1_EL1, Interrupt Controller Virtual Highest iority Pe ng Inter pt Register 1
- ICV_IAR0_EL1, Interrupt Controller Virtual Interru Acknow dge R 0
- ICV_IAR1_EL1, Interrupt Controller Virtual Inter t Ackn edge Regi er 1
- ICV_IGRPEN0_EL1, Interrupt Controller Virtual Inte Group 0 E able Register
- ICV_IGRPEN1_EL1, Interrupt Controller V Interru Group Enable Register
- ICV_NMIAR1_EL1, Interrupt Controlle irtual n-mask terrupt Acknowledge Register 1
- ICV_PMR_EL1, Interrupt Controller V tual Inte pt Priority Mask Register
- ICV_RPR_EL1, Interrupt Controller Vi l R ning y Register

The above registers are as described in [

$R_{GWZVL}$ If FEAT_GCIE_LEGACY is im mented and CH_VC LR_EL2.V3 is 0, the registers listed in rule WSWHT are UNDEFINED.

$I_{YFTVZ}$ If FEAT_GCIE_LEGACY is implem ted, accesses to any of the corresponding ICC_ registers listed in rule WSWHT are UNDEF D.

---

**Note**

This also mea s that a esses to any of the registers listed in rule WSWHT at EL2 or EL3 are UNDEFINED.

---

$R_{XKKDD}$ When the FEAT_GICv3 or FEAT_GCIE_LEGACY is implemented, the registers listed in Table 9.230 are present.

When FEAT_ CIE_LEGACY is implemented, all of the following are true:

- When Legacy operation is enabled, all of the following are true:
  - When any of the following are true, accesses at EL1 to the registers listed in Table 9.230 generate a Trap exception taken to EL2 and reported using EC syndrome value 0x18:
    - ICH_HCR_EL2.TC is 1.
    - HCR_EL2.IMO is 1.
    - HCR_EL2.FMO is 1.
  - Otherwise, when all of the following are true, accesses at EL1 to the registers listed in Table 9.230 are UNDEFINED:
    - ICH_HCR_EL2.TC is 0.
    - HCR_EL2.IMO is 0.
    - HCR_EL2.FMO is 0.
  - Accesses at any Exception level other than EL1 to the registers listed in Table 9.230 are UNDEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- When Legacy operation is disabled, the registers listed in Table 9.230 are UNDEFINED at any Exception level.

**Table 9.230: GICv3 CPU Interface registers**

| Register |
|---|
| ICC_SGI0R_EL1 |
| ICC_SGI1R_EL1 |
| ICC_ASGI1R_EL1 |

The registers listed in Table 9.230 are described in [3].

$R_{SLVXV}$ When FEAT_GCIE and FEAT_EL2 are implemented, all of the following are true:

- The Effective value of ICC_SRE_EL3.Enable is 1.
- The Effective value of ICC_SRE_EL3.SRE is 1.

$R_{MCTMX}$ When FEAT_GCIE and FEAT_EL2 are implemented, all of the following are true:

- ICC_SRE_EL1 is present.
- Accesses to the following fields are RAO/WI:
  - ICC_SRE_EL1.SRE.
  - ICC_SRE_EL1.DFB.
  - ICC_SRE_EL1.DIB.

The layout and fields of ICC_SRE_EL1 are described in [3].

$R_{HQBBN}$ When FEAT_GCIE and FEAT_EL2 are implemented, all of the following are true:

- ICC_SRE_EL2 is present.
- Accesses to the following fields are RAO/WI:
  - ICC_SRE_EL2.SRE.
  - ICC_SRE_EL2.DFB.
  - ICC_SRE_EL2.DIB.
  - ICC_SRE_EL2.Enable.

The layout and fields of ICC_SRE_EL2 are described in [3].

$R_{PWDWZ}$ When either FEAT_GCIE_LEGACY or FEAT_GCIE_LEGACY is implemented, the following fields are present:

- HFGRTR_EL2.ICC_IGRPENn_EL1
- HFGWTR_EL2.ICC_IGRPENn_EL1

The above fields are described in [1].

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter 10
# Registers and memory maps

RETIRED

This chapter describes the registers and memory mapped interfaces of GICv5 components.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 10.1 Memory-mapped programmer's model

$R_{JYKRN}$ Accesses to registers that are not implemented is CONSTRAINED UNPREDICTABLE and results in either:

- The access has RAZ/WI behavior.
- The location has the same behavior as RES0.

$D_{JGRJR}$ The access type definitions for the memory-mapped register interface are:

- **RAZ/WI**: Read-As-Zero, Writes Ignored.
- **RO**: Read only. Writes are ignored.
- **RW**: Read and write.
- **RES0**: RES0 as defined in [1].
- **RES1**: RES1 as defined in [1].
- **W1C**: Write-One-to-Clear. Writes of 0 to the bit are ignored. A write of 1 clears the bit to 0.
- **W1S**: Write-One-to-Set. Writes of 0 to the bit are ignored. A write of 1 sets the bit to 1.
- **WI**: Write ignore. Reads return an UNKNOWN value.
- **WO**: Write only. Reads return an UNKNOWN value.
- **WO/UNKNOWN**: The same as **WO**.
- **UNKNOWN/WI**: The same as **WI**.

$R_{DQMPD}$ In the register definitions, references to other registers are always to a register in the same register frame as where the access is made.

$R_{HQTXF}$ All registers for the IWB, the ITS, and IRS are little-endian.

$R_{WRLMJ}$ All registers are either 32-bit or 64-bit.

An implementation supports aligned 32-bit accesses to all registers and aligned 64-bit accesses to 64-bit registers. When a 32-bit access occurs to a 64-bit register, bits[63:32] of the register are accessed at offset +4 and bits[31:0] at offset 0.

All aligned single-copy atomic accesses to a register of the same size as the access are treated as single-copy atomic accesses by the GIC.

Aligned 16-bit half-word accesses are supported only to the lower half of the following registers:

- ITS_TRANSLATER
- ITS_RL_TRANSLATER

All other accesses are *illegal accesses*.

$R_{VXDYH}$ The behavior of an illegal access is CONSTRAINED UNPREDICTABLE to one of the following:

- The access is RAZ/WI.
- The access completes on the GIC and one of the following is true:
  - The access is a read and the read returns an UNKNOWN value.
  - The access is a write and the write sets any field of the accessed register, including fields outside the access, to an UNKNOWN value. It is CONSTRAINED UNPREDICTABLE whether side effects of a write occur or not.
- The access generates an abort.

A 64-bit access to two adjacent 32-bit registers is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The access is RAZ/WI.
- A read returns the value of both registers and a write updates both registers, as though two 32-bit accesses were performed in an UNPREDICTABLE order.
- One of the pair of registers is read or written and the other register is RAZ/WI, as though a single 32-bit access was performed to an UNPREDICTABLE one of the pair of registers.
- The access generates an abort.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$D_{RFLQG}$ Some register fields are *Guarded* by another field in the same or another register. In the GICv5 specification, a Guarded register field is **RO** or **WI** unless the field by which it is Guarded is in a certain state.

For example, IRS_IST_BASER.ADDR is Guarded by IRS_IST_BASER.VALID and IRS_PE_SELR.IAFFID is Guarded by IRS_PE_STATUSR.IDLE.

$R_{BRRWQ}$ Writing to Guarded register fields that are **RO** or **WI** have no effects beyond optionally reporting an error in the software error reporting mechanism.

$I_{LPSXY}$ Updating the value of a Guarded field requires a separate write to clear the condition that causes the Guarded field to be **RO** or **WI** before the Guarded field can be updated.

For example, if IRS_IST_BASER.VALID is 1, IRS_IST_BASER.ADDR is **RO** and a write to IRS_IST_BASER that sets IRS_IST_BASER.VALID to 0 does not have any effect on the value of IRS_IST_BASER.ADDR. Updating IRS_IST_BASER.ADDR can be done using the following sequence:

1. Performing a write that sets IRS_IST_BASER.VALID to 0.
2. Performing a write that sets IRS_IST_BASER.ADDR to a new value.

$S_{MXQGP}$ Software must ensure proper ordering when updating Guarded fields.

For a write that updates the field by which other fields are Guarded, all of the following are true:

- In order to update the Guarded fields, software must ensure that the GIC has observed the write before it observes a write to the Guarded fields.
- This ordering can be enforced through the use of an appropriate memory type for accesses and by issuing appropriate barriers, or by reading back the write before making another write to the Guarded fields.

For a write that updates Guarded fields, all of the following are true:

- To make sure the GIC observes the write to the Guarded fields, software must ensure that the GIC has observed the write to the Guarded fields before it observes a write to the field by which they are Guarded.
- This ordering can be enforced through the use of an appropriate memory type for accesses and by issuing appropriate barriers, or by reading back the value written to the Guarded fields before making another write to the field by which they are Guarded.

$D_{DYZRK}$ Some registers have write side effects when a *WO* field is set to 1.

Registers with such write side effects are tracked with an IDLE bit in a status register to indicate when the write side effects are complete.

$I_{TWSRH}$ Following a write that sets a *WO* field to 1, when the corresponding IDLE bit is 1, another write that sets the the *WO* field to 1, without any intervening write that sets the field to 0, will cause the write side effects.

$I_{XXQBX}$ If a write occurs to a register that has write side effects when a *WO* field is set to 1, and if the field is not set to 1, the write side effects will not take place and no other registers are updated as a result of the write.

For example, if a write occurs to IRS_VPE_SELR that does not set IRS_VPE_SELR.S to 1, writes to any of the IRS_VPE_x registers apply to the VPE selected the last time a write occurred that set IRS_VPE_SELR.S to 1.

$S_{FFYYH}$ 64-bit registers that have write side effects, and support 32-bit accesses, may define a *WO* field that causes the side effects when set to 1 in either of the 32-bit halves.

Software can specify the full 64-bit register value as input to the side effects of the write using 32-bit accesses by using a sequence such as the following:

1. Write to the 32-bit half of the register which does not contain the *WO* field.
2. Ensure the write is observed before a following write by using a barrier instruction or reading back the value written to the 32-bit half of the register.
3. Write to the other 32-bit half of the register which contains the *WO* field and set the field to 1.

$R_{SCDGF}$ Registers are not required to support being the target of exclusive or atomic read-modify-write update operations.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{VBHBM}$ Register frames associated with an Interrupt Domain are accessible via the PAS associated with that Interrupt Domain.

For register frames not associated with the MPPAS, except for the ITS_TRANSLATE_FRAME and IRS_SETLPI_FRAME, it is IMPLEMENTATION DEFINED whether the register frames are also accessible in the MPPAS at the same addresses.

$I_{WJTJR}$ Any memory-mapped access to a GICv5 register region is defined to be beyond the PE.

$S_{DGVJR}$ Armv9 does not require the size of each element accessed by a multi-register load or store instruction to be identifiable by the memory system beyond the PE.

Software can use a Device-nGRE or stronger memory-type, and use only single register load and store instructions, to create memory accesses that are supported by a GICv5 implementation.

Reads and writes of the memory-mapped registers complete in the order in which they arrive at the GIC.

For accesses to different register locations, software can determine the order in which they are arrive at the GIC by doing all of the following:

- Accessing the GIC using the Device-nGnRnE or Device-nGnRE memory types.
- Using the appropriate memory barriers.

Software can determine the completion of a write by doing one of the following:

- Accessing the GIC using the Device-nGnRnE memory type and executing a DSB barrier.
- Reading back the value written.

For more information on memory types and barriers ensuring completion, see [1].

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 10.2 IRS register frames

$I_{DLZJM}$ For each Interrupt Domain, the IRS exposes the following register frames:

- The IRS configuration frame.
- Optionally, the SETLPI register frame.

Each register frame size is 64K.

$I_{JWWVY}$ In the IRS register definitions, references to other registers are always to a register in the register frame for the same Interrupt Domain on the same IRS as where the access is made.

$I_{GGXDW}$ If IRS_IDR0.SETLPI is 0, the SETLPI register frame is not present.

### 10.2.1 IRS_CONFIG_FRAME

The IRS_CONFIG_FRAME characteristics are:

#### Purpose

Contains control registers for an IRS for an Interrupt Domai

An IRS configuration register frame is present for each s ported I errupt Domain on each IRS.

This register frame is accessible in the PAS associat with t nterrupt D main.

It is IMPLEMENTATION DEFINED whether this gister fr me is als essible in the MPPAS at the same address.

The base address is distinct from the base ad ess of GIC register frame, including the configuration register frames for other Interrupt Dom ns on a SS.

The base address is aligned to 64K

#### Attributes

IRS_CONFIG_FRAME is a bl k of size: 0x10000 bytes

#### Contents

| Offset | Name |
|---|---|
| 0x0000 | IRS_IDR0 |
| 0x0004 | IRS_IDR1 |
| 0x0008 | IRS_IDR2 |
| 0x000C | IRS_IDR3 |
| 0x0010 | IRS_IDR4 |
| 0x0014 | IRS_IDR5 |
| 0x0018 | IRS_IDR6 |
| 0x001C | IRS_IDR7 |
| 0x0040 | IRS_IIDR |
| 0x0044 | IRS_AIDR |
| 0x0080 | IRS_CR0 |
| 0x0084 | IRS_CR1 |
| 0x00C0 | IRS_SYNCR |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Offset | Name |
|---|---|
| 0x00C4 | IRS_SYNC_STATUSR |
| 0x0100 | IRS_SPI_VMR |
| 0x0108 | IRS_SPI_SELR |
| 0x010C | IRS_SPI_DOMAINR |
| 0x0110 | IRS_SPI_RESAMPLER |
| 0x0114 | IRS_SPI_CFGR |
| 0x0118 | IRS_SPI_STATUSR |
| 0x0140 | IRS_PE_SELR |
| 0x0144 | IRS_PE_STATUSR |
| 0x014 | RS_PE_CR |
| 0 180 | IRS_IST_BASER |
| 0190 | IRS_IST_CFGR |
| 0x0 | RS_IST_STATUSR |
| 0x01C0 | IRS_MAP_L2_ISTR |
| | IRS_VMT_BASER |
| 0x0210 | IRS_VMT_CFGR |
| 0x0214 | IRS_VMT_STATUSR |
| 0x0240 | IRS_VPE_SELR |
| 0x0248 | IRS_VPE_DBR |
| 0x0250 | IRS_VPE_HPPIR |
| 0x0258 | IRS_VPE_CR0 |
| 0x025C | IRS_VPE_STATUSR |
| 0x0280 | IRS_VM_DBR |
| 0x0288 | IRS_VM_SELR |
| 0x028C | IRS_VM_STATUSR |
| 0x02C0 | IRS_VMAP_L2_VMTR |
| 0x02C8 | IRS_VMAP_VMR |
| 0x02D0 | IRS_VMAP_VISTR |
| 0x02D8 | IRS_VMAP_L2_VISTR |
| 0x02E0 | IRS_VMAP_VPER |
| 0x0300 | IRS_SAVE_VMR |
| 0x0308 | IRS_SAVE_VM_STATUSR |
| 0x0340 | IRS_MEC_IDR |
| 0x0344 | IRS_MEC_MECID_R |
| 0x0380 | IRS_MPAM_IDR |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| Offset | Name |
|---|---|
| 0x0384 | IRS_MPAM_PARTID_R |
| 0x03C0 | IRS_SWERR_STATUSR |
| 0x03C8 | IRS_SWERR_SYNDROMER0 |
| 0x03D0 | IRS_SWERR_SYNDROMER1 |
| 0XE00 – ↪0XEFC | IMPLEMENTATION DEFINED |

Direct accesses to other offsets in this block are RAZ/WI.

**Note**

For information on the use of offsets 0xE00 - 0xEFC see *10.6 CoreSight identification registers*.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.2.1.1 IRS_AIDR

The IRS_AIDR characteristics are:

**Purpose**

IRS Architecture Identification Register. Identifies the GIC architecture version to which the implementation conforms.

**Configuration**

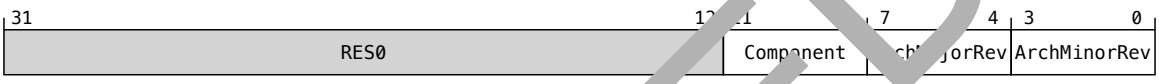
There are no configuration notes.

**Attributes**

IRS_AIDR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:12]**

Reserved, RES0.

**Component, bits [11:8]**

GIC component

| Component | Meaning |
|---|---|
| 0b0000 | IRS |
| 0b0001 | ITS |
| 0b0010 | IWB |

**ArchMajorRev, bits [7:4]**

Major Architecture revision.

| ArchMajorRev | Meaning |
|---|---|
| 0b0000 | GICv5.x |

**ArchMinorRev, bits [3:0]**

Minor Architecture revision.

| ArchMinorRev | Meaning |
|---|---|
| 0b0000 | GICv5.0 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_AIDR

Accesses to this register use the following encodings:

Accessible at offset `0x0044` from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.2 IRS_CR0

The IRS_CR0 characteristics are:

**Purpose**

IRS control register 0.

**Configuration**

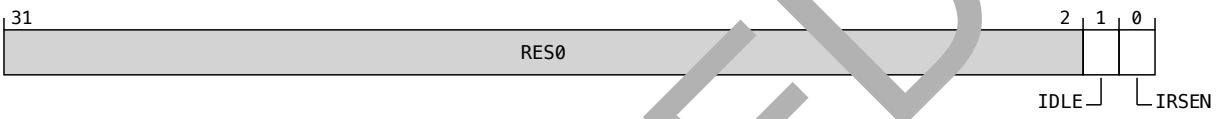
There are no configuration notes.

**Attributes**

IRS_CR0 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**IDLE, bit [1]**

Whether the transition between enabled and disabled states of the IRS for the Interrupt Domain is complete.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of updating IRSEN are not guaranteed to have completed. |
| 0b1 | The effects of updating IRSEN are have completed. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Accesses to this field is RO.

**IRSEN, bit [0]**

Controls whether the IRS Forwards and Recalls candidate HPPIs to PEs for the Interrupt Domain.

This field is Guarded by IRS_CR0.IDLE.

| IRSEN | Meaning |
|---|---|
| 0b0 | The IRS is disabled for the Interrupt Domain. |
| 0b1 | The IRS is enabled for the Interrupt Domain. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When IRS_CR0.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

#### *Accessing IRS_CR0*

Accesses to this register use the following encodings:

Accessible at offset `0x0080` from IRS_CONFIG_FRAME

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.3 IRS_CR1

The IRS_CR1 characteristics are:

**Purpose**

IRS configuration register 1

**Configuration**

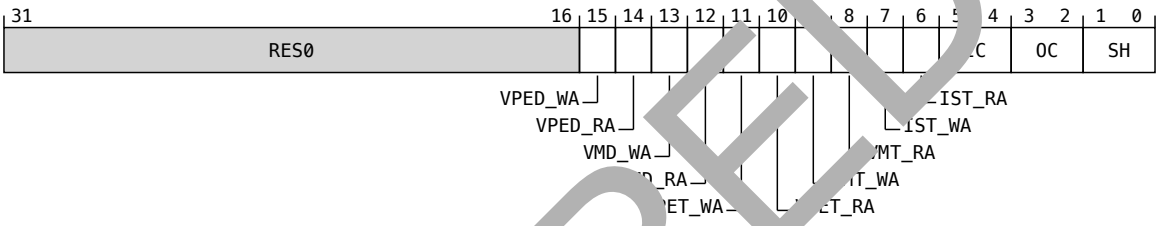
There are no configuration notes.

**Attributes**

IRS_CR1 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**VPED_WA, bit [15]**

**When IRS_IDR0.VIRT ==**

Write-Allocate hint for the VPE descriptors.

| VPED_WA | Meaning |
|---|---|
| 0b0 | No Write-Allocate. |
| 0b1 | Write-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VPED_RA, bit [14]**

**When IRS_IDR0.VIRT == 1:**

Read-Allocate hint for the VPE descriptors.

| VPED_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VMD_WA, bit [13]**

**When IRS_IDR0.VIRT == 1:**

Write-Allocate hint for the VM descriptors.

| VMD_WA | Meaning |
|---|---|
| 0b0 | No Write-Allocate. |
| 0b1 | Write-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VMD_RA, bit [12]**

**When IRS_IDR0.VIRT == 1:**

Read-Allocate hint for the VM descriptors.

| VMD_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VPET_WA, bit [11]**

**When IRS_IDR0.VIRT == 1:**

Write-Allocate hint for the VPE table.

| VPET_WA | Meaning |
|---|---|
| 0b0 | No Write-Allocate. |
| 0b1 | Write-Allocate. |

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VPET_RA, bit [10]**

**When IRS_IDR0.VIRT == 1:**

Read-Allocate hint for the VPE table.

| VPET_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VMT_WA, bit [9]**

**When IRS_IDR0.VIRT == 1:**

Write-Allocate hint for the VMT table.

| VMT_WA | Meaning |
|---|---|
| 0b0 | No Write-Allocate. |
| 0b1 | Write-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VMT_RA, bit [8]**

**When IRS_IDR0.VIRT == 1:**

Read-Allocate hint for the VM table.

| VMT_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Otherwise:**

Reserved, RES0.

**IST_WA, bit [7]**

Write-Allocate hint for ISTs.

When IRS_IDR0.VIRT is 1, this control applies to the virtual ISTs as well as the physical IST.

| IST_WA | Meaning |
|---|---|
| 0b0 | No Write-Allocate. |
| 0b1 | Write-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**IST_RA, bit [6]**

Read-Allocate hint for ISTs.

When IRS_IDR0.VIRT is 1, this control applies to the virtual ISTs as well as the physical IST.

| IST_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**IC, bits [5:4]**

Controls the Inner Cacheability attribute used when the IRS accesses tables as a requester.

| IC | Meaning |
|---|---|
| 0b00 | Non-cacheable. |
| 0b01 | Write-Back Cacheable. |
| 0b10 | Write-Through Cacheable. |
| 0b11 | Reserved, treated as 0b00. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**OC, bits [3:2]**

Controls the Outer Cacheability attribute used when the IRS accesses tables as a requester.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| OC | Meaning |
|---|---|
| 0b00 | Non-cacheable. |
| 0b01 | Write-Back Cacheable. |
| 0b10 | Write-Through Cacheable. |
| 0b11 | Reserved, treated as 0b00. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**SH, bits [1:0]**

Controls the Shareability attribute used when the IRS accesses tables as a requester.

| SH | Meaning |
|---|---|
| 0b00 | Non-shareable. |
| 0b01 | Reserved, treated as 0b00. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

When IRS_CR1.OC is 0b00 and IRS_CR1.IC is 0b00, this field is IGNORED and behaves as Outer Shareable.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing IRS_CR1***

Accesses to this register use the following encodings:

Accessible at offset 0x0004 from IRS_CONFIG_FRAME

- When IRS_IST_BASER.VALID == 1 or IRS_IST_STATUSR.IDLE == 0, accesses to this register are RO.
- When IRS_VMT_BASER.VALID == 1 or IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.2.1.4 IRS_IDR0

The IRS_IDR0 characteristics are:

**Purpose**

IRS identification register 0. Contains read-only fields with information about the IRS GIC component.

**Configuration**

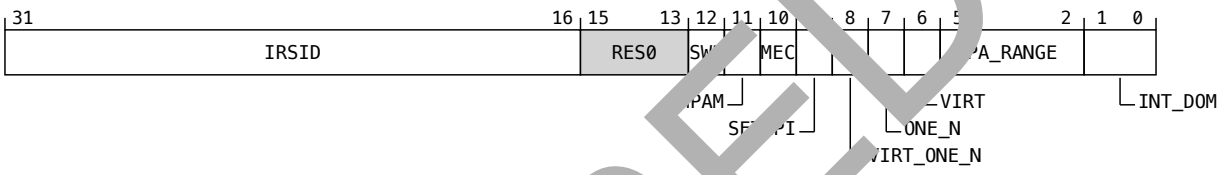
There are no configuration notes.

**Attributes**

IRS_IDR0 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**IRSID, bits [31:16]**

Unique identifier for this IRS in the syst

This value is the same across all Int upt Do ins for this IRS.

**Bits [15:13]**

Reserved, RES0.

**SWE, bit [12]**

Software err reportin support in the Interrupt Domain.

| SWE | Meaning |
|---|---|
| 0b0 | Software error reporting is not supported. |
| 0b1 | Software error reporting is supported. |

Support r Software error reporting is optional.

**MPAM, bit [11]**

Memory Partitioning And Monitoring (MPAM) support.

| MPAM | Meaning |
|---|---|
| 0b0 | MPAM is not supported. |
| 0b1 | MPAM is supported. |

Support for MPAM is optional.

**MEC, bit [10]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When IRS_IDR0.INT_DOM == 0b11:**

Support for Memory Encryption Contexts (MEC) for the Realm Interrupt Domain.

| MEC | Meaning |
|---|---|
| 0b0 | Memory Encryption Contexts are not supported. |
| 0b1 | Memory Encryption Contexts are supported. |

Support for MEC is optional.

**Otherwise:**

Reserved, RES0.

**SETLPI, bit [9]**

**When IRS_IDR2.LPI == 1:**

Whether the IRS implements the IRS SETLPI register.

The IRS SETLPI register allows an MSI to set a physical LPI Pending without translation by an ITS.

| SETLPI | Meaning |
|---|---|
| 0b0 | The IRS does not implement the IRS SETLPI register. |
| 0b1 | The IRS implements the IRS SETLPI register. |

**Otherwise:**

Reserved, RES0.

**VIRT_ONE_N, bit [8]**

Whether virtual 1ofN is supported.

| VIRT_ONE_N | Meaning |
|---|---|
| 0b0 | Virtual 1ofN is not supported. |
| 0b1 | Virtual 1ofN is supported. |

This field is RES0, if any of the following are true:

- Virt is 0.
- ONE_N is 0.

**ONE_N, bit [7]**

| ONE_N | Meaning |
|---|---|
| 0b0 | 1ofN is not supported. |
| 0b1 | 1ofN is supported. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**VIRT, bit [6]**

This field is RES0 for the EL3 Interrupt Domain

| VIRT | Meaning |
|---|---|
| 0b0 | Virtualization is not supported. |
| 0b1 | Virtualization is supported. |

**PA_RANGE, bits [5:2]**

Physical Address range supported.

The physical address range corresponds to the system physical addr[illegible]s size.

The value of this field is the same for all Interrupt Domains acr[illegible] all IRSs in th[illegible] system.

| [illegible]_RANGE | Meaning |
|---|---|
| 0b000[illegible] | 32 bits, 4GB |
| 0b[illegible]001 | 36 bits, 64GB |
| 0b0010 | 40 bits, 1TB |
| 0b[illegible]011 | 42 bits, 4TB |
| 0b0100 | 44 bits, 16TB |
| 0b0101 | 48 bits, 256TB |
| 0b0110 | 52 bits, 4PB |
| 0b0111 | 56 bits, 64PB |

Values not d[illegible]ned [illegible]ove are r[illegible]served.

**INT_DO[illegible], bits [1:0]**

[illegible]he Interr[illegible]pt Domain [illegible]at the register frame containing this register controls.

| INT_DOM | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### *Accessing IRS_IDR0*

Accesses to this register use the following encodings:

Accessible at offset 0x0000 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.5 IRS_IDR1

The IRS_IDR1 characteristics are:

**Purpose**

IRS identification register 1. Contains read-only fields with information about the IRS GIC component.

**Configuration**

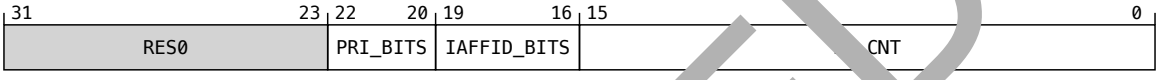
There are no configuration notes.

**Attributes**

IRS_IDR1 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:23]**

Reserved, RES0.

**PRI_BITS, bits [22:20]**

The number of priority bits implemented, minus one.

| PRI_BITS | Meaning |
|---|---|
| 0b000 | 1 bit of priority. |
| 0b001 | 2 bits of priority. |
| 0b010 | 3 bits of priority. |
| 0b011 | 4 bits of priority. |
| 0b100 | 5 bits of priority. |

Values not defined above are reserved.

When fewer than 5 bits are implemented, the lower order priority bits are RES0.

**IAFFID_BITS, bits [19:16]**

Number of bits of IAFFID supported - 1.

Unimplemented upper bits of IAFFID are RES0 when sending and receiving commands between a PE and the IRS.

**PE_CNT, bits [15:0]**

The number of PEs connected to this IRS.

***Accessing IRS_IDR1***

Accesses to this register use the following encodings:

Accessible at offset 0x0004 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.6 IRS_IDR2

The IRS_IDR2 characteristics are:

**Purpose**

IRS identification register 2. Contains read-only fields with information about the IRS GIC component.

**Configuration**

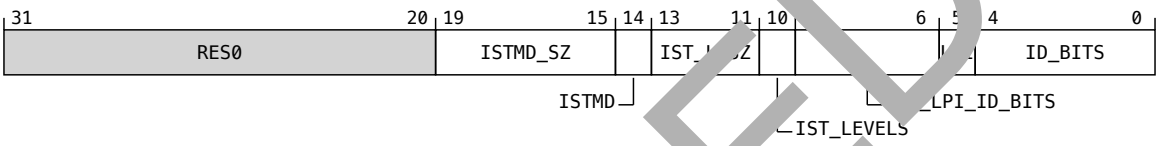
There are no configuration notes.

**Attributes**

IRS_IDR2 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:20]**

Reserved, RES0.

**ISTMD_SZ, bits [19:15]**

Describes the minimum number of INTID.ID bits which requires a level 2 ISTE size of 16 bytes to store metadata.

When the configured number of INTID.ID bits is smaller than ISTMD_SZ, the minimum required size of a level 2 ISTE is 8 bytes.

When the configured number of INTID.ID bits is larger than or equal to ISTMD_SZ, the minimum required size of a level 2 ISTE is 16 bytes.

When ISTMD is 0, this field is RES0.

**ISTMD, bit [14]**

Whether the IST entries contain metadata storage.

When the IST entires contain metadata storage, the size of a level 2 ISTE is 8 bytes or 16 bytes.

The size of an IST entry containing metadata depends on the configured ID range and the values reported in ISTMD_SZ.

| ISTMD | Meaning |
|---|---|
| 0b0 | The IST entries do not require storage for metadata and the size of a level 2 ISTE is 4 bytes. |
| 0b1 | The IST entries require storage for metadata and the size of a level 2 ISTE is 8 bytes or 16 bytes. |

**IST_L2SZ, bits [13:11]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When IRS_IDR2.IST_LEVELS == 1:**

Supported level 2 IST sizes when a 2-level IST structure is used.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| IST_L2SZ | Meaning |
|---|---|
| 0b001 | Level 2 IST sizes supported: 4KB |
| 0b010 | Level 2 IST sizes supported: 16KB |
| 0b011 | Level 2 IST sizes supported: 4KB and 16KB |
| 0b100 | Level 2 IST sizes supported: 64KB |
| 0b101 | Level 2 IST sizes supported: 4KB and 64KB |
| 0b110 | Level 2 IST sizes supported: 16KB and 64KB |
| 0b111 | Level 2 IST sizes supported: 4KB, 16KB, and 64KB |

Values not defined above are reserved.

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**IST_LEVELS, bit [10]**

**When IRS_IDR2.LPI == 1 or IRS_IDR0.VIRT == 1:**

Whether a 2-level structure is supported for the physical and virtual ISTs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

| IST_LEVELS | Meaning |
|---|---|
| 0b0 | Only a single level linear structure is supported. |
| 0b1 | 2-level structure is supported in addition to the linear structure. |

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**MIN_LPI_ID_BITS, bits [9:6]**

**When IRS_IDR2.LPI == 1 or IRS_IDR0.VIRT == 1:**

The minimum number of LPI_ID_BITS supported.

The maximum value supported for this field is 14.

When physical LPIs are not supported, this field reports the minimal supported number of virtual LPI ID bits that can be configured for each VM.

This field has an IMPLEMENTATION DEFINED value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Access to this field is RO.

**Otherwise:**

Reserved, RES0.

**LPI, bit [5]**

Whether physical LPIs are implemented.

When physical LPIs are not supported, the physical IST registers are not implemented.

The value reported in this field has no impact on the support for virtual LPIs.

**ID_BITS, bits [4:0]**

The maximum number of INTID.ID bits that the IRS supports.

The maximum supported value of this field is 24.

### *Accessing IRS_IDR2*

Accesses to this register use the following encodings:

Accessible at offset `0x0008` from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.7 IRS_IDR3

The IRS_IDR3 characteristics are:

**Purpose**

IRS identification register 3. Contains read-only fields with information about the IRS GIC component.

**Configuration**

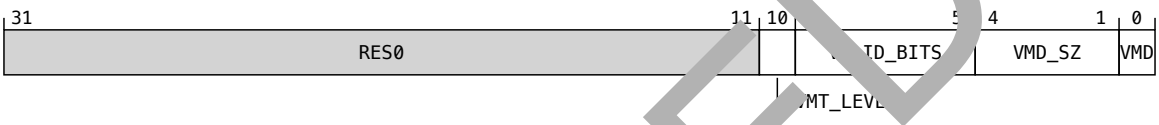
There are no configuration notes.

**Attributes**

IRS_IDR3 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:11]**

Reserved, RES0.

**VMT_LEVELS, bit [10]**

**When IRS_IDR0.VIRT == 1:**

Levels supported for the VM table.

| VMT_LEVELS | Meaning |
|---|---|
| 0b0 | Only a single level linear structure is supported. |
| 0b1 | 2-level structure is supported in addition to the linear structure. |

**Otherwise:**

Reserved, RES0.

**VM_ID_BITS, bits [9:5]**

**When IRS_IDR0.VIRT == 1:**

The maximum number of VM ID bits that the IRS supports.

The minimum supported value of this field is 8 and the maximum supported value is 16.

**Otherwise:**

Reserved, RES0.

**VMD_SZ, bits [4:1]**

**When IRS_IDR0.VIRT == 1:**

Specifies the size in bytes of a VM descriptor.

Each VM descriptor is 2^(VMD_SZ) bytes.

The minimum valid value for this field is 3 and the maximum valid value is 12.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

This corresponds to a minimum descriptor size of 8 bytes and a maximum descriptor size of 4096 bytes.

**Otherwise:**

Reserved, RES0.

**VMD, bit [0]**

**When IRS_IDR0.VIRT == 1:**

Whether each VM requires an IMPLEMENTATION DEFINED memory area.

| VMD | Meaning |
|---|---|
| 0b0 | The VMs do not require VM descriptor areas. |
| 0b1 | Each VM requires a VM descriptor area. |

**Otherwise:**

Reserved, RES0.

### *Accessing IRS_IDR3*

Accesses to this register use the following encoding:

Accessible at offset 0x000C from IRS_CONFIG_FRAME:

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.8 IRS_IDR4

The IRS_IDR4 characteristics are:

**Purpose**

IRS identification register 4. Contains read-only fields with information about the IRS GIC component.

**Configuration**

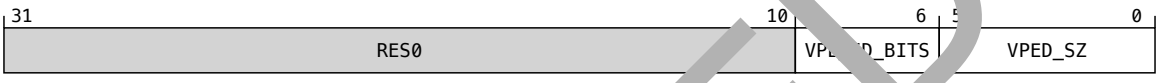
There are no configuration notes.

**Attributes**

IRS_IDR4 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:10]**

Reserved, RES0.

**VPE_ID_BITS, bits [9:6]**

The number of VPE ID bits supported - 1.

The minimum supported value of this field is 7.

Accessing this field has the following behavior:

- When IRS_IDR0.VIRT == 0, access to this field is RES0.
- Otherwise, access to this field is RO.

**VPED_SZ, bits [5:0]**

Specifies the size in bytes of a VPE Descriptor.

Each VPE Descriptor is 2^(VPED_SZ) bytes.

The minimum valid value for this field is 3 and the maximum valid value is 12.

This corresponds to a minimum descriptor size of 8 bytes and a maximum descriptor size of 4096 bytes.

Accessing this field has the following behavior:

- When IRS_IDR0.VIRT == 0, access to this field is RES0.
- Otherwise, access to this field is RO.

***Accessing IRS_IDR4***

Accesses to this register use the following encodings:

Accessible at offset 0x0010 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.9 IRS_IDR5

The IRS_IDR5 characteristics are:

**Purpose**

IRS identification register 5. Contains read-only fields with information about the IRS GIC component.

**Configuration**

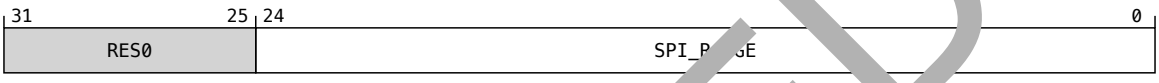
There are no configuration notes.

**Attributes**

IRS_IDR5 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:25]**

Reserved, RES0.

**SPI_RANGE, bits [24:0]**

The total number of SPIs in the system.

The maximum value reported by this field is 2^24-1.

The number reported in this register is the same across all Interrupt Domains and across all IRSs in a system.

***Accessing IRS_IDR5***

Accesses to this register use the following encodings:

Accessible at offset 0x0014 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.10 IRS_IDR6

The IRS_IDR6 characteristics are:

**Purpose**

IRS identification register 6. Contains read-only fields with information about the range of SPI INTIDs managed by this IRS.

**Configuration**

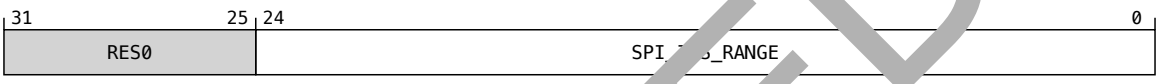
There are no configuration notes.

**Attributes**

IRS_IDR6 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:25]**

Reserved, RES0.

**SPI_IRS_RANGE, bits [24:0]**

The number of SPI INTID.ID managed on this IRS.

The value reported in this field is less than or equal to the value reported in IRS_IDR5.SPI_RANGE.

If IRS_IDR5.SPI_RANGE is 0, this field is RES0.

***Accessing IRS_IDR6***

Accessible at offset 0x0080 from IRS_CONFIG_FRAME

Accesses to this register are RO

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.11 IRS_IDR7

The IRS_IDR7 characteristics are:

**Purpose**

IRS identification register 7. Contains read-only fields with information about the minimum SPI INTID.ID value implemented on this IRS.

**Configuration**

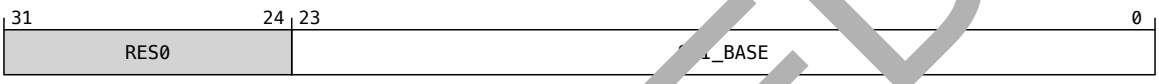
There are no configuration notes.

**Attributes**

IRS_IDR7 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:24]**

Reserved, RES0.

**SPI_BASE, bits [23:0]**

The minimum SPI INTID.ID implemented on this IRS.

If IRS_IDR5.SPI_RANGE is 0 or IRS_IDR6.SPI_IRS_RANGE is 0, this field is RES0.

***Accessing IRS_IDR7***

Accesses to this register use the following encodings:

Accessible at offset 0x001C from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.12 IRS_IIDR

The IRS_IIDR characteristics are:

**Purpose**

IRS Implementer Identification Register. Provides information about the implementation and implementer of the GIC, and architecture version supported.

**Configuration**

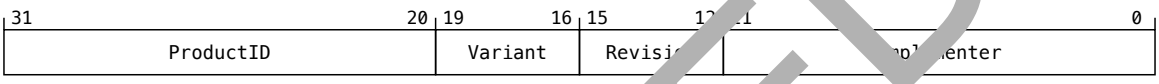
There are no configuration notes.

**Attributes**

IRS_IIDR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**ProductID, bits [31:20]**

IMPLEMENTATION DEFINED value identifying the GIC part

When the IRS_PIDR0-IRS_PIDR1 registers are present, Arm expects that the IRS_PIDR0.PART_0 and IRS_PIDR1.PART_1 fields match the value of IRS_IIDR.ProductID.

If required, however, an implementation is permitted to provide values for the IRS_PIDR0.PART_0 and IRS_PIDR1.PART_1 fields that do not match the value of IRS_IIDR.ProductID

**Variant, bits [19:16]**

IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product

**Revision, bits [15:12]**

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product

**Implementer, bits [11:**

Contains the JEP106 code of the company that implemented the GIC

For an Arm implementation, the JEP106 code is 0x43B

When the CoreSight ID registers are implemented, Arm expects that the JEP106 identification code in the IRS_PIDR1-IRS_PIDR4 registers matches that reported in IRS_IIDR.

***Accessing IRS_IIDR***

Accesses to this register use the following encodings:

Accessible at offset `0x0040` from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 10.2.1.13 IRS_IST_BASER

The IRS_IST_BASER characteristics are:

**Purpose**

IRS IST base address register

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_IST_STATUSR.IDLE is 1.

**Attributes**

IRS_IST_BASER is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:6]**

Bits[55:6] of the base physical address of the IST.

This field is Gu[illegible]ed by IRS_IST_[illegible]SER.VALID.

When IRS_IST_CFGR.STRUCTURE is 0, ADDR points to a linear IST and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N depends on the ISTSZ and LPI_ID_BITS fields in IRS_IST_CFGR as follows:

  N = Max(5, (ISTSZ + 1) + LPI_ID_BITS).

- This means that the level 2 ISTE array is aligned to the size of the array or to a 64-byte boundary when its size is smaller than 64 bytes.

When IRS_IST_CFGR.STRUCTURE is 1, ADDR points to the level 1 table in a 2-level IST and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N depends on L2SZ, ISTSZ, and LPI_ID_BITS in IRS_IST_CFGR as follows:

  N = Max(5, LPI_ID_BITS - ((10 - ISTSZ) + (2 * L2SZ)) + 2)

- This means that the level 1 IST is aligned to the size of the level 1 table or to a 64-byte boundary when its size is smaller than 64 bytes.

See 4.7 *The interrupt state table (IST)* for more information.

Access to any level of the IST and any additional memory accesses occurring as a result of the address in this field are performed using the PAS of the Interrupt Domain where this register is accessed.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_IST_BASER.VALID == 1, access to this field is RO.
- Otherwise, access to this field is RW.

**Bits [5:1]**

Reserved, RES0.

**VALID, bit [0]**

Whether the ADDR points to a valid IST.

| VALID | Meaning |
|---|---|
| 0b0 | The IST address is not valid. |
| 0b1 | The IST address is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to [illegible].

### *Accessing IRS_IST_BASER*

Accesses to this register use the following encoding:

Accessible at offset 0x0180 from IRS_CONFIG_FRAME

- When IRS_IDR2.LPI == 0, accesses to this register are RAZ/WI.
- When IRS_IST_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.14 IRS_IST_CFGR

The IRS_IST_CFGR characteristics are:

**Purpose**

IRS IST configuration register

**Configuration**

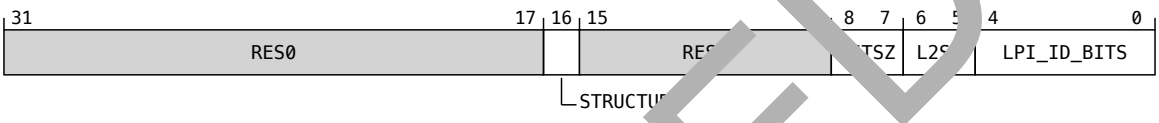
There are no configuration notes.

**Attributes**

IRS_IST_CFGR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:17]**

Reserved, RES0.

**STRUCTURE, bit [16]**

Whether the IST uses a linear or 2-level structure.

| STRUCTURE | Meaning |
|---|---|
| 0b0 | A linear IST structure is used. |
| 0b1 | A 2-level IST structure is used. |

When IRS_IDR2.IST_LEVELS is 0, this field is RES0.

The reset behavior of this field is:

On a GIC reset, this field resets to an UNKNOWN value.

**Bits [15:9]**

Reserved, RES0.

**ISTSZ, bits [8:7]**

The size of each level 2 ISTE.

If this field is programmed to specify a size smaller than the minimum required size or programmed to a reserved value, it is treated as having the value corresponding to the minimum required size for all other purposes than a direct read of the register.

See 4.7 *The interrupt state table (IST)* for more information about the minimum required size.

| ISTSZ | Meaning |
|---|---|
| 0b00 | The size of a level 2 ISTE is 4 bytes. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| ISTSZ | Meaning |
|---|---|
| 0b01 | The size of a level 2 ISTE is 8 bytes. |
| 0b10 | The size of a level 2 ISTE is 16 bytes. |

Values not defined above are reserved.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**L2SZ, bits [6:5]**

Level 2 IST size when a 2-level IST structure is used.

| L2SZ | Meaning |
|---|---|
| 0b00 | The level 2 IST size is 4KB. |
| 0b01 | The level 2 IST size is 16KB. |
| 0b10 | The level 2 IST size is 64KB. |

Values not defined above are reserved.

IRS_IDR2.IST_L2SZ reports the supported values.

If LPI_ID_BITS <= ((10 - ISTSZ) + (2 * L2SZ)) and STRUCTURE is 1, all of the following are true:

- The IST consists of a single L1_ISTE and a single level 2 IST.
- The level 2 IST contains (2^LPI_ID_BITS) entries.
- The IRS Domain is allowed to access the full level 2 IST size as specified by L2SZ.

Arm recommends that STRUCTURE is 0 when LPI_ID_BITS <= ((10 - ISTSZ) + (2 * L2SZ)).

See 4.7 *The interrupt state table (IST)* for more information.

If programming a reserved value or an unsupported value, the IRS Domain behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

When STRUCTURE is 0, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**LPI_ID_BITS, bits [4:0]**

The number of LPIs for the IRS Domain.

The IST contains 2^(LPI_ID_BITS) level 2 IST entries in total.

The minimum value for this field is IRS_IDR2.MIN_LPI_ID_BITS.

If programmed to a value smaller than the minimum, the field is treated as having the minimum value for all other purposes than reading back the field.

The maximum value for this field is IRS_IDR2.ID_BITS.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than a direct read of the register. The maximum value is reported in IRS_IDR2.ID_BITS.

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an UNKNOWN value.

#### Accessing IRS_IST_CFGR

Accesses to this register use the following encodings:

Accessible at offset `0x0190` from IRS_CONFIG_FRAME

- When IRS_IDR2.LPI == 0, accesses to this register are RAZ/WI.
- When IRS_IST_BASER.Valid == 1 or IRS_IST_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.15 IRS_IST_STATUSR

The IRS_IST_STATUSR characteristics are:

**Purpose**

IRS physical IST management status register.

**Configuration**

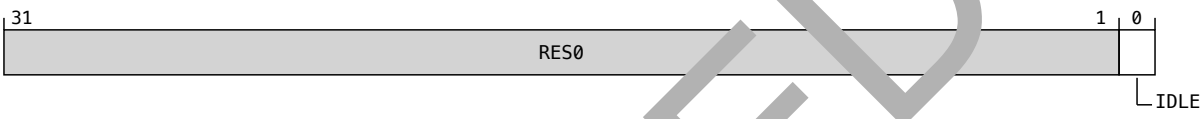
There are no configuration notes.

**Attributes**

IRS_IST_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Whether the effects of any of the following writes are complete:

- A write that updates IRS_IST_BASER.VALID.
- A write to IRS_MAP_ISTR.ID.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the write are not complete. |
| 0b1 | The effects of the write are complete. |

The reset behavior of this field is:

On a GIC reset, this field resets to '1'.

Access to this field is RO.

***Accessing IRS_IST_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x0194 from IRS_CONFIG_FRAME

- When IRS_IDR2.LPI == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.16 IRS_MAP_L2_ISTR

The IRS_MAP_L2_ISTR characteristics are:

**Purpose**

IRS map physical level 2 IST register

**Configuration**

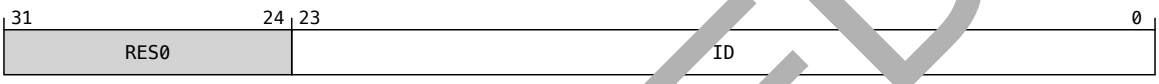
The effects of a write to this register are not guaranteed to have completed before IRS_IST_STATUSR.IDLE is 1.

**Attributes**

IRS_MAP_L2_ISTR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




A write to this register makes the specified physical level 2 IST valid.

There are no effects of a write to this register if any of the following are true:

- The physical IST is invalid.
- The physical IST uses a linear structure.
- The LPI INTID.ID is outside the configured physical LPI range.
- The level 2 IST is already valid.

**Bits [31:24]**

Reserved, RES0.

**ID, bits [23:0]**

An LPI INTID.ID covered by the level 1 ISTE corresponding to the level 2 IST that is made valid.

If unimplemented upper bits of the INTID.ID are not zero, it is IMPLEMENTATION DEFINED whether the upper bits are treated as 0 or the interrupt message is ignored by the IRS.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

***Accessing IRS_MAP_L2_ISTR***

Accesses to this register use the following encodings:

Accessible at offset `0x01C0` from IRS_CONFIG_FRAME

- When IRS_IDR2.LPI == 0, accesses to this register are RAZ/WI.
- When IRS_IST_STATUSR.IDLE == 0, accesses to this register are UNKNOWN/WI.
- When IRS_IST_BASER.VALID == 0, accesses to this register are UNKNOWN/WI.
- When IRS_IST_CFGR.STRUCTURE == 0, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.17 IRS_MEC_IDR

The IRS_MEC_IDR characteristics are:

**Purpose**

IRS MEC identification register. Contains read-only fields with information about the IRS support for MEC.

**Configuration**

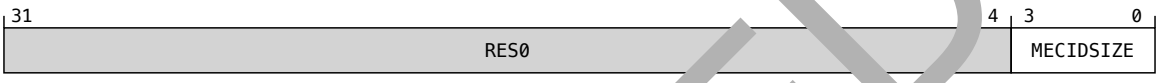
There are no configuration notes.

**Attributes**

IRS_MEC_IDR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:4]**

Reserved, RES0.

**MECIDSIZE, bits [3:0]**

**When IRS_IDR0.MEC == 1:**

The number of bits minus one of MECID supported by the IRS.

The maximum permitted value is 0xF, which indicates a MECID width of 16 bits.

The value 0x0 is a valid encoding and indicates that one bit of MECID is supported.

**Otherwise:**

Reserved, RES0.

***Accessing IRS_MEC_IDR***

Accesses to this register use the following encodings:

Accessible at offset 0x0340 from IRS_CONFIG_FRAME

- When IRS_IDR0.INT_DOM != 0b11, accesses to this register are RAZ/WI.
- When IRS_IDR0.MEC != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.18 IRS_MEC_MECID_R

The IRS_MEC_MECID_R characteristics are:

**Purpose**

IRS MEC MECID register for the Realm PAS.

**Configuration**

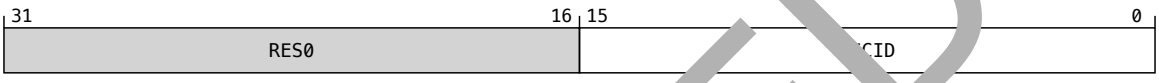
There are no configuration notes.

**Attributes**

IRS_MEC_MECID_R is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**MECID, bits [15:0]**

MECID for IRS access to Realm PA spac[illegible]

- Accesses to physical and [illegible]T entri
- Accesses to VM table [illegible]tries and V[illegible]E table [illegible]trie.
- Accesses to VM de[illegible]ors and VPE [illegible]scriptors.

Bits above the supported MEC[illegible]size, indicated in IRS_MEC_IDR.MECIDSIZE are RES0.

If MECIDSIZE is less than 0xF, the [illegible]S treats bits [15:MECIDSIZE+1] of this field as zero.

The reset b[illegible]avior of [illegible]is field is:

- On a GIC [illegible]t, this fiel[illegible]esets to an UNKNOWN value.

***Access[illegible]ng IR[illegible]_MEC_[illegible]EC[illegible]_R***

Acc[illegible]es to th[illegible]ister use the following encodings:

Accessi[illegible]at offset 0x0344 from IRS_CONFIG_FRAME

- When I[illegible]_IDR0.INT_DOM != 0b11, accesses to this register are RAZ/WI.
- When IRS_IDR0.MEC != 1, accesses to this register are RAZ/WI.
- When IRS_IST_BASER.VALID == 1 or IRS_IST_STATUSR.IDLE == 0, accesses to this register are RO.
- When IRS_VMT_BASER.VALID == 1 or IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.19 IRS_MPAM_IDR

The IRS_MPAM_IDR characteristics are:

**Purpose**

IRS MPAM identification register. Contains read-only fields with information about the IRS support for MPAM.

**Configuration**

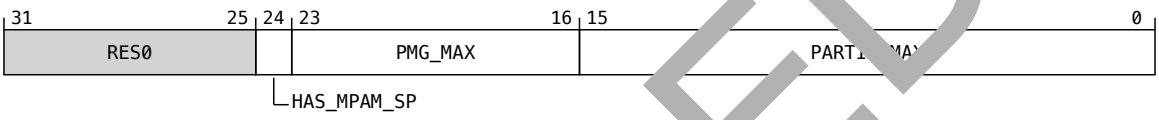
There are no configuration notes.

**Attributes**

IRS_MPAM_IDR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:25]**

Reserved, RES0.

**HAS_MPAM_SP, bit [24]**

Whether the IRS has support for MPAM PARTID space selection for the Interrupt Domain.

If HAS_MPAM_SP is 1, the IRS uses the MPAM PARTID specified by IRS_MPAM_PARTID_R.MPAM_SP.

If HAS_MPAM_SP is 0, the following PARTID space is used for IRS accesses to memory:

- Accesses made for the Secure Interrupt Domain use the Secure PARTID space.
- Accesses made for the Non-secure Interrupt Domain use the Non-secure PARTID space.
- Accesses made for the Root Interrupt Domain use the Root or Secure PARTID space.
- Accesses made for the Realm Interrupt Domain use the Realm PARTID space.

The value of this field is the same across all Interrupt Domains for an IRS.

**PMG_MAX, bits [23:16]**

The maximum PMG value that is permitted to be used for the IRS for the Interrupt Domain.

The PMG bit width is defined as the bit position of the most significant 1 in PMG_MAX[7:0], plus one, or is defined as zero if PMG_MAX is zero.

For example, if PMG_MAX == `0x0f`, the PMG bit width is 4.

This field is permitted to be zero-sized.

**PARTID_MAX, bits [15:0]**

The maximum PARTID value that is permitted to be used for the IRS for the Interrupt Domain.

The PARTID bit width is defined as the bit position of the most significant 1 in PARTID_MAX[15:0], plus one, or is defined as zero if PARTID_MAX is zero.

For example, if PARTID_MAX == `0x0034`, the PARTID bit width is 6.

This field is permitted to be zero-sized, but Arm recommends that it is non-zero when MPAM is implemented.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_MPAM_IDR

Accesses to this register use the following encodings:

Accessible at offset `0x0380` from IRS_CONFIG_FRAME

- When IRS_IDR0.MPAM != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.20 IRS_MPAM_PARTID_R

The IRS_MPAM_PARTID_R characteristics are:

**Purpose**

IRS MPAM PARTID and PMG register.

**Configuration**

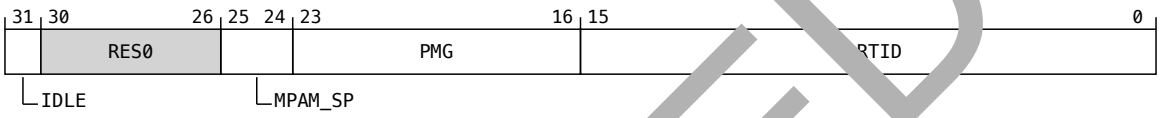
There are no configuration notes.

**Attributes**

IRS_MPAM_PARTID_R is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**IDLE, bit [31]**

Whether the effects of the previous write to this register are complete.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the previous write to this register are not complete. |
| 0b1 | The effects of the previous write to this register are complete. |

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Access to this field is RO.

**Bits [30:26]**

Reserved, RES0

**MPAM_SP, bits [25:24]**

**When IRS_MPAM_IDR.HAS_MPAM_SP == 1 and IRS_IDR0.INT_DOM == 0b00:**

MPAM PARTID space for the IRS for the Secure Interrupt Domain.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b00 | Secure PARTID space. |
| 0b01 | Non-secure PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the IRS using an UNKNOWN PARTID space.

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**When IRS_MPAM_IDR.HAS_MPAM_SP == 1 and IRS_IDR0.INT_DOM == 0b01:**

MPAM PARTID space for the IRS for the Non-secure Interrupt Domain.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b01 | Non-secure PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the IRS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**When IRS_MPAM_IDR.HAS_MPAM_SP == 1 and IRS_IDR0.INT_DOM == 0b10:**

MPAM PARTID space for the IRS for the EL3 Interrupt Domain.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b00 | Secure PARTID space. |
| 0b01 | Non-secure PARTID space. |
| 0b10 | Root PARTID space. |
| 0b11 | Realm PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the IRS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**When IRS_MPAM_IDR.HAS_MPAM_SP == 1 and IRS_IDR0.INT_DOM == 0b11:**

MPAM PARTID space for the IRS for the Realm Interrupt Domain.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| MPAM_SP | Meaning |
|---|---|
| 0b01 | Non-secure PARTID space. |
| 0b11 | Realm PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the IRS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**Otherwise:**

Reserved, RES0.

**PMG, bits [23:16]**

PMG for accesses to memory by the IRS for the Interrupt Domain.

Bits above the supported PMG bit width, as indicated by IRS_MPAM_IDR.PMG_MAX, are RES0.

If a value greater than IRS_MPAM_IDR.PMG_MAX is programmed, an UNKNOWN PMG is used.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0x00.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**PARTID, bits [15:0]**

PARTID for accesses to memory by the IRS for the Interrupt Domain.

Bits above the supported PARTID bit width, as indicated by IRS_MPAM_IDR.PARTID_MAX, are RES0.

If a value greater than IRS_MPAM_IDR.PARTID_MAX is programmed, an UNKNOWN PARTID is used.

This field is Guarded by IRS_MPAM_PARTID_R.IDLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0x0000.

Accessing this field has the following behavior:

- When IRS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

### *Accessing IRS_MPAM_PARTID_R*

Accesses to this register use the following encodings:

Accessible at offset 0x0384 from IRS_CONFIG_FRAME

- When IRS_IDR0.MPAM != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.21 IRS_PE_CR0

The IRS_PE_CR0 characteristics are:

**Purpose**

IRS PE control register 0.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_PE_STATUSR.IDLE is 1.

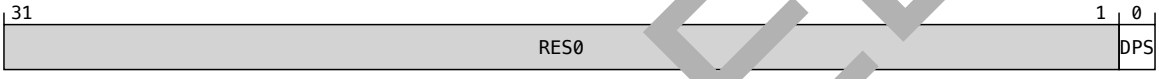
Following a write to this register, IRS_PE_STATUSR.V is updated.

**Attributes**

IRS_PE_CR0 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




The fields in this register provide access to the configuration of the PE specified in the last write to IRS_PE_SELR.

The PE configuration applies to the IRS Domain corresponding to the IRS configuration register frame that this register is accessed in.

**Bits [31:1]**

Reserved, RES0.

**DPS, bit [0]**

**When IRS_IDR0.ONE_N == 1:**

Disable 1ofN PE selection.

| DPS | Meaning |
|---|---|
| 0b0 | 1ofN PE selection is enabled. An interrupt configured to use the 1ofN Routing mode is permitted to select this PE. |
| 0b1 | 1ofN PE selection is disabled. An interrupt configured to use the 1ofN Routing mode is not permitted to select this PE. |

This field determines whether the selected PE is permitted to be selected for 1ofN interrupt delivery.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_PE_CR0

Accesses to this register use the following encodings:

Accessible at offset `0x0148` from IRS_CONFIG_FRAME

- When [IRS_PE_STATUSR.V, IRS_PE_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.22 IRS_PE_SELR

The IRS_PE_SELR characteristics are:

**Purpose**

IRS PE selection register.

**Configuration**

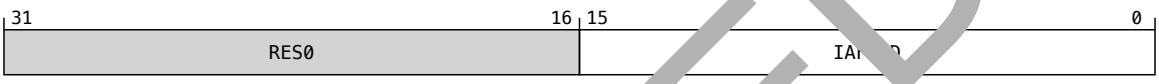
The effects of a write to this register are not guaranteed to have completed before IRS_PE_STATUSR.IDLE is 1.

**Attributes**

IRS_PE_SELR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**IAFFID, bits [15:0]**

PE interrupt Affinity ID.

Selects a PE whose configuration can be accessed via IRS_PE_CR0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing IRS_PE_SELR***

Accesses to this register use the following encodings:

Accessible at offset 0x0040 from IRS_CONFIG_FRAME

- When IRS_PE_STATUSR.IDLE == 0, accesses to this register are WI.
Otherwise accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.23 IRS_PE_STATUSR

The IRS_PE_STATUSR characteristics are:

**Purpose**

IRS PE status register

**Configuration**

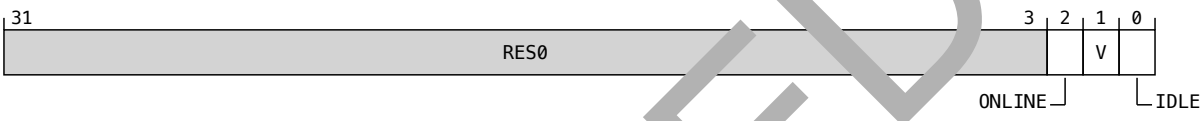
There are no configuration notes.

**Attributes**

IRS_PE_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




The fields in this register return information about the PE specified in the last write to IRS_PE_SELR and the status of the last write to IRS_PE_CR0.

**Bits [31:3]**

Reserved, RES0.

**ONLINE, bit [2]**

Whether the PE is online or offline.

When the IRS determines that there is a candidate HPPI for the PE and the PE is offline the IRS generates a Wake Request to the PE.

When the IRS determines that there is a candidate HPPI for the PE and the PE is online the IRS Forwards the candidate HPPI to the PE.

When IDLE is not '0b0', the value of this field is UNKNOWN.

| ONLINE | Meaning |
|---|---|
| 0b0 | The PE is offline. |
| 0b1 | The PE is online. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**V, bit [1]**

Whether the value last written to IRS_PE_SELR selected a valid PE.

When IDLE is 0, the value of this field is UNKNOWN.

| V | Meaning |
|---|---|
| 0b0 | The PE selected using IRS_PE_SELR is not valid. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| V | Meaning |
|---|---|
| 0b1 | The PE selected using IRS_PE_SELR is valid. |

This field resets to 0 to indicate that there was no write to IRS_PE_SELR that selected a valid PE since the IRS was reset.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

**IDLE, bit [0]**

Following a write to IRS_PE_SELR, when this field returns 1 and V[1] read from IRS_PE_CR0 returns the configuration value for the PE when the accesses are performed on the same IRS.

Following a write to IRS_PE_CR0, when this field returns 1, the effects of the write have completed.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of writing to IRS_PE_SELR and IRS_PE_CR0 are not guaranteed to be complete. |
| 0b1 | The effects of writing to IRS_PE_SELR and IRS_PE_CR0 are complete. |

The reset behavior of this field is:

- On a GIC reset, this field resets to

***Accessing IRS_PE_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x0144 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.24 IRS_SAVE_VMR

The IRS_SAVE_VMR characteristics are:

**Purpose**

IRS save VM register

**Configuration**

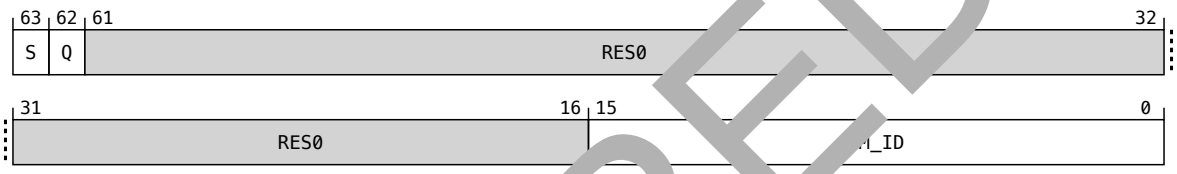
The effects of a write to this register are not guaranteed to have completed before IRS_SAVE_VM_STATUSR.IDLE is 1.

**Attributes**

IRS_SAVE_VMR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**S, bit [63]**

Writing 1 to this field initiates the operation to save the state and configuration of virtual interrupts for the VM identified by VM_ID to the virtual ISTs.

The operation is complete when IRS_SAVE_VMSTATUSR.IDLE is 1. At that point, IRS_SAVE_VMSTATUSR.Q indicates whether the VM remained in the Quiesced state throughout the operation.

| S | Meaning |
|---|---|
| 0b0 | The write has no effect on the virtual ISTs. |
| 0b1 | The write saves the state of virtual interrupts to the virtual ISTs. |

Access to this field is WO/UNKNOWN.

**Q, bit [62]**

Writing 1 to this field queries whether the VM specified by VM_ID is Quiesced on all IRSs since the last write that set S to 1 and returns the result in IRS_SAVE_VM_STATUSR.Q when IRS_SAVE_VM_STATUSR.IDLE is 1.

When a write sets S to 1, the value written to this field is IGNORED.

Following a write that updates VM_ID, if there has been no write that set S to 1, the value returned in IRS_SAVE_VM_STATUSR.Q is UNKNOWN.

| Q | Meaning |
|---|---|
| 0b0 | The write has no effect on the value returned in IRS_SAVE_VM_STATUSR.Q. |
| 0b1 | The write queries whether the VM is Quiesced on all IRSs since the last write that set S to 1. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Access to this field is WO/UNKNOWN.

**Bits [61:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

The VM ID specifying the VM whose virtual interrupt state should be written to the ISTs.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

#### *Accessing IRS_SAVE_VMR*

Accesses to this register use the following encodings:

Accessible at offset `0x0300` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_SAVE_VM_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.25 IRS_SAVE_VM_STATUSR

The IRS_SAVE_VM_STATUSR characteristics are:

**Purpose**

IRS save VM status register.

**Configuration**

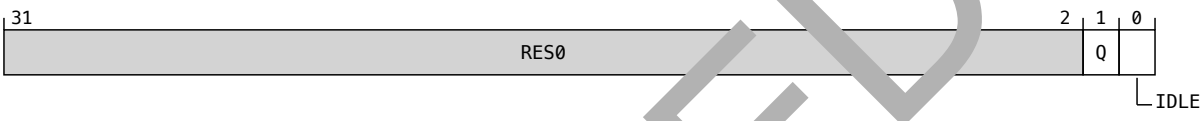
There are no configuration notes.

**Attributes**

IRS_SAVE_VM_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




The fields in this register return information about the last write to IRS_SAVE_VMR.

**Bits [31:2]**

Reserved, RES0.

**Q, bit [1]**

Reports whether the VM specified by IRS_SAVE_VMR.VM_ID was in the Quiesced state at the time of the most recent write to IRS_SAVE_VMR.S that set it to 1, and whether it remained in the Quiesced state until one of the following occurs:

- The operation that saves the state and configuration of virtual interrupts to the virtual ISTs is complete and IDLE is 1.
- A write sets IRS_SAVE_VMR.Q to 1 and IDLE is 1.

This field is UNKNOWN, if any of the following are true:

- From the time the value in IRS_SAVE_VMR.VM_ID was last changed, there has not been a write that set IRS_SAVE_VMR.S to 1.
- The VM specifies an invalid VM.
- IDLE is 0.

See 'Save and restore of virtual interrupts' for more information.

| Q | Meaning |
|---|---|
| 0b0 | The VM is not Quiesced. |
| 0b1 | The VM is Quiesced. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**IDLE, bit [0]**

Reports the status of the last write to IRS_SAVE_VMR.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the last write to IRS_SAVE_VMR are not guaranteed to be complete. |
| 0b1 | The effects of the last write to IRS_SAVE_VMR are complete. |

This field resets to 1 to allow initial writes to registers Guarded by this field. Because there has been no write at reset, this does not imply that any invalidate operation is complete.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

#### *Accessing IRS_SAVE_VM_STATUSR*

Accesses to this register use the following encodings:

Accessible at offset 0x0308 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.26 IRS_SPI_CFGR

The IRS_SPI_CFGR characteristics are:

**Purpose**

IRS SPI configuration register.

Allows software to read and update the configuration of the SPI selected by IRS_SPI_SELR.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_SPI_STATUSR.IDLE is 1.

Following a write to this register, IRS_SPI_STATUSR.V is updated.

**Attributes**

IRS_SPI_CFGR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register is updated after a write to IRS_SPI_SELR when IRS_SPI_STATUSR.{V, IDLE} is {1, 1}.

**Bits [31:1]**

Reserved, RES0.

**TM, bit [0]**

The Trigger mode of the SPI.

| TM | Meaning |
|---|---|
| 0b0 | Edge-triggered |
| 0b1 | Level-sensitive |

It is IMPLEMENTATION DEFINED whether access to this field for the selected SPI is **RW** or **RO**.

Note: This field controls the type of event generated by SPI signals. This is a separate control from the INTID Handling mode.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing IRS_SPI_CFGR***

Accesses to this register use the following encodings:

Accessible at offset 0x0114 from IRS_CONFIG_FRAME

- When [IRS_SPI_STATUSR.V, IRS_SPI_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.27 IRS_SPI_DOMAINR

The IRS_SPI_DOMAINR characteristics are:

**Purpose**

IRS SPI Interrupt Domain configuration Register.

Allows software to read and update the Interrupt Domain assignment of the SPI selected by IRS_SPI_SELR.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_SPI_STATUSR.IDLE is 1.

Following a write to this register, IRS_SPI_STATUSR.V is updated.

**Attributes**

IRS_SPI_DOMAINR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register is updated after a write to IRS_S_ SEL_ wh_ _SPI_STATUSR.{V, IDLE} is {1, 1}.

**Bits [31:2]**

Reserved, RES0.

**DOMAIN, bits [1:0]**

Configures the Int_rupt Domain_ sociated with the SPI.

Some SPIs m_y be statically assigned to a Domain, in which case this field always returns the statically assigned D_main for_e SPI.

To check if a S_ an be dyn_mically assigned to a Domain, software must read back the value in this field aft_ atte_pting a_ updat_ to establish if the update was successful once IRS_SPI_STATUSR.IDLE is 1.

| DOMAIN | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

Programming an Interrupt Domain not supported by the IRS results in CONSTRAINED UNPREDICTABLE behavior with the following options:

- The SPI behaves as if it is not assigned to any Interrupt Domain.
- The SPI is treated as being assigned to another supported Interrupt Domain for all other purposes than reading back this field

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_SPI_DOMAINR

Accesses to this register use the following encodings:

Accessible at offset `0x010C` from IRS_CONFIG_FRAME

- When IRS_IDR0.INT_DOM != 0b10, accesses to this register are RAZ/WI.
- When [IRS_SPI_STATUSR.V, IRS_SPI_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.28 IRS_SPI_RESAMPLER

The IRS_SPI_RESAMPLER characteristics are:

**Purpose**

IRS SPI resample register.

Resample an SPI signal.

**Configuration**

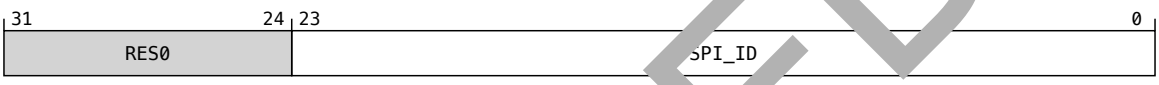
There are no configuration notes.

**Attributes**

IRS_SPI_RESAMPLER is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register allows resampling an SPI.

**Bits [31:24]**

Reserved, RES0.

**SPI_ID, bits [23:0]**

SPI INTID.ID of the SPI to resample.

Following a write to this register if all of the following are true, the SPI is resampled:

- The SPI is managed on this IRS.
- The SPI can be accessed in this IRS Domain.

Otherwise, a write to this register has no effect.

See 'Physical SPIs' for more information.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing IRS_SPI_RESAMPLER***

Accesses to this register use the following encodings:

Accessible at offset `0x0110` from IRS_CONFIG_FRAME

Accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.29 IRS_SPI_SELR

The IRS_SPI_SELR characteristics are:

**Purpose**

IRS SPI selection register.

**Configuration**

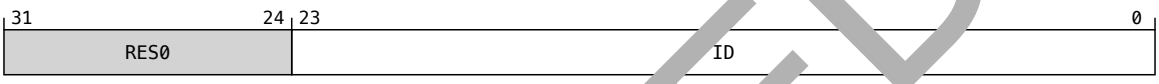
The effects of a write to this register are not guaranteed to have completed before IRS_SPI_STATUSR.IDLE is 1.

**Attributes**

IRS_SPI_SELR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:24]**

Reserved, RES0.

**ID, bits [23:0]**

Selects the SPI that the following registers access:

- IRS_SPI_CFGR
- IRS_SPI_DOMAINR
- IRS_SPI_VMR

Only implemented SPIs with INTID in the range from IRS_IDR7.SPI_BASE to (IRS_IDR7.SPI_BASE + IRS_IDR6.SPI_IRS_RANGE - 1) may be selected.

The reset behavior of this field is:

- On GIC reset, this field resets to an UNKNOWN value.

***Accessing IRS_SPI_SELR***

Accesses to this register use the following encodings:

Accessible at offset 0x0108 from IRS_CONFIG_FRAME

- When IRS_SPI_STATUSR.IDLE == 0, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.30 IRS_SPI_STATUSR

The IRS_SPI_STATUSR characteristics are:

**Purpose**

IRS SPI status register.

**Configuration**

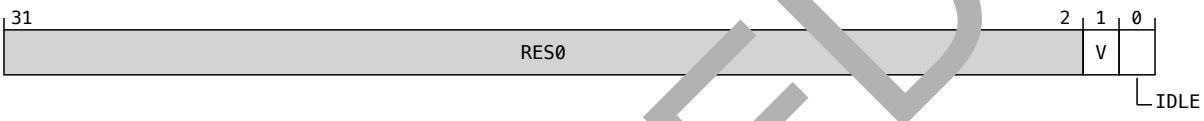
There are no configuration notes.

**Attributes**

IRS_SPI_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**V, bit [1]**

Whether the value last written to IRS_SPI_SELR selects a valid SPI that is managed on this IRS and can be accessed in this IRS Domain.

This field is updated if and only if a write occurs to any of the following registers in this IRS Domain:

- IRS_SPI_CFGR
- IRS_SPI_SELR
- IRS_SPI_VMR

Following a write to any of the above registers, if this field returns 0, the write to the register has no effects other than updating the value in this register.

SPIs with INTID.IID in the range described by IRS_IDR7.SPI_BASE and IRS_IDR6.SPI_IRS_RANGE are managed by this IRS.

When IRS_SPI_SELR specifies an SPI and all of the following are true, this field returns 1:

- The selected SPI is implemented.
- The selected SPI is managed by this IRS.
- The selected SPI is assigned to this IRS Domain or this IRS Domain is the EL3 IRS Domain.

Otherwise, this field returns 0.

When IDLE is 0, the value of this field is UNKNOWN.

| V | Meaning |
|---|---|
| 0b0 | The SPI selected using IRS_SPI_SELR is not a valid SPI that is managed on this IRS and can be accessed in this IRS Domain. |
| 0b1 | The SPI selected using IRS_SPI_SELR is a valid SPI that is managed on this IRS and can be accessed in this IRS Domain. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

This field resets to 0 to indicate that there was no write to IRS_SPI_SELR that selected a valid SPI that is managed on this IRS since the IRS was reset.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0'`.

**IDLE, bit [0]**

Whether writes to any of the following registers have completed:

- IRS_SPI_CFGR
- IRS_SPI_DOMAINR
- IRS_SPI_SELR
- IRS_SPI_VMR

Following a write to IRS_SPI_SELR, when {IDLE, V} is {1, 1} reads from any of the other registers returns the configuration value for the SPI.

Following a write to any of the registers listed above, on this IRS, when {IDLE, V} is {1, 1}, the effects of the write are complete.

| IDLE | Meaning |
|---|---|
| `0b0` | The effects of writing to the SPI selection and configuration registers are not guaranteed to have completed. |
| `0b1` | The effects of writing to the SPI selection and configuration registers have completed. |

This field resets to 1 to allow initial writes to registers Guarded by this field. Because there has been no write at reset, this does not imply that any invalidate operation is complete.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'1'`.

***Accessing IRS_SPI_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset `0x0144` from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.31 IRS_SPI_VMR

The IRS_SPI_VMR characteristics are:

**Purpose**

IRS SPI VM assignment register.

Allows software to read and update the VM assignment of the SPI selected by 'IRS_SPI_SELR'.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_SPI_STATUSR.IDLE is 1.

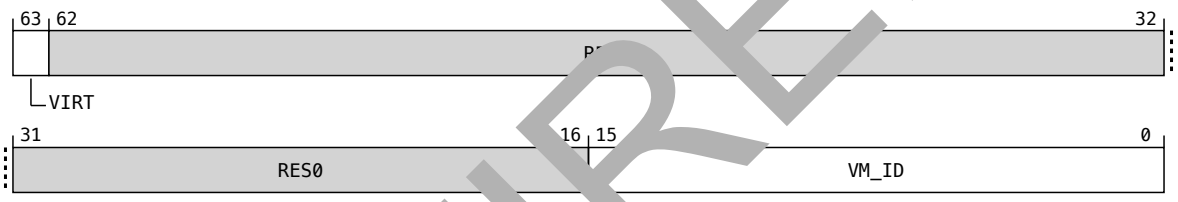
Following a write to this register, IRS_SPI_STATUSR.V is updated.

**Attributes**

IRS_SPI_VMR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register is updated after a write to IRS_SPI_SELR when IRS_SPI_STATUSR.{V, IDLE} is {1, 1}.

**VIRT, bit [63]**

Whether the SPI selected by IRS_SPI_SELR is assigned as a virtual SPI to the VM.

| VIRT | Meaning |
|---|---|
| 0b0 | The SPI is not assigned to a VM. |
| 0b1 | The SPI is assigned to the VM specified by VM_ID. |

**Bits [62:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

Identifies the VM.

On a read, when VIRT is 0, the value of this field is UNKNOWN.

On a write, if the existing value of VIRT is 1, the write to this field is IGNORED.

***Accessing IRS_SPI_VMR***

When the value of VIRT in the register is 1, meaning that the selected SPI is assigned to a VM, any write that does not set VIRT to 0 is ignored.

To assign an SPI to a different VM, the SPI must first be unassigned from the old VM, and then subsequently assigned to the new VM.

Accesses to this register use the following encodings:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Accessible at offset `0x0100` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When [IRS_SPI_STATUSR.V, IRS_SPI_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- When !IsSPIDVISupported(IRS_SPI_SELR.ID), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 10.2.1.32 IRS_SWERR_STATUSR

The IRS_SWERR_STATUSR characteristics are:

**Purpose**

IRS software error status register. Specifies whether a software error has been reported. If an error is reported, it contains syndrome information for the error.

**Configuration**

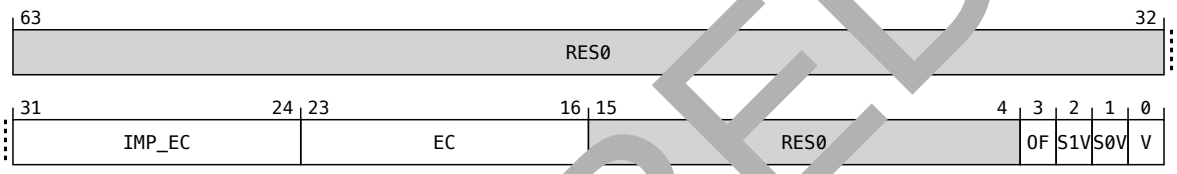
There are no configuration notes.

**Attributes**

IRS_SWERR_STATUSR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:32]**

Reserved, RES0.

**IMP_EC, bits [31:24]**

IMPLEMENTATION DEFINED error code when IRS_SWERR_STATUS.EC == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - IRS_SWERR_STATUSR.V == 0
  - IRS_SWERR_STATUSR.EC != 0
- Otherwise access to this field is RO.

**EC, bits [23:16]**

Specifies the error code that software can use to triage and handle the error.

| EC | Meaning |
|---|---|
| 0b0 | An error was reported because of an IMPLEMENTATION DEFINED reason. |
| 0b1 | Failed access to L1_ISTE due to an external abort. |
| 0b10 | Failed access to L2_ISTE due to an external abort. |
| 0b11 | Failed access to L1_VMTE due to an external abort. |
| 0b100 | Failed access to L2_VMTE due to an external abort. |
| 0b101 | Failed access to VPE Table due to an external abort. |
| 0b110 | Failed access to VPE descriptor due to an external abort. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| EC | Meaning |
|---|---|
| 0b111 | Failed access to VM descriptor due to an external abort. |
| 0b1000 | A physical LPI could not be processed because IRS_IST_BASER.VALID is 0. |
| 0b1001 | A physical LPI could not be processed because L1_ISTE.VALID is 0. |
| 0b1010 | A physical LPI was not processed because the LPI INTID.ID > (2 ^ IRS_IST_CFGR.LPI_ID_BITS) - 1. |
| 0b1011 | A virtual interrupt could not be processed because L1_ISTE.VALID is 0 |
| 0b1100 | A virtual interrupt could not be processed because L1_VMTE.VALID is 0. |
| 0b1101 | A virtual interrupt could not be processed because L2_VMTE.VALID is 0. |
| 0b1110 | A virtual LPI was not processed because L2_VMTE.LPI_ISTE.VALID is 0. |
| 0b1111 | A virtual SPI was not processed because L2_VMTE.SPI_ISTE.VALID is 0. |
| 0b10000 | A virtual LPI was not processed because the LPI INTID.ID > (2 ^ L2_VMTE.LPI_ID_BITS) - 1. |
| 0b10001 | A virtual SPI was not processed because the SPI INTID.ID > (2 ^ L2_VMTE.SPI_ID_BITS) |
| 0b10010 | A virtual LPI signaled by an ITS was not processed because the VM ID > (2 ^ IRS_VMT_CFGR.VM_ID_BITS) - 1 |
| 0b10011 | A virtual interrupt was not processed because the VPE ID > (2 ^ L2_VMTE.VPE_ID_BITS) - 1. |
| 0b10100 | A virtual interrupt was signaled in a non-EL3 Interrupt Domain by an ITS, or via the GIC CPENDR system instruction, and the interrupt was unreachable because IRS_IDR0.VMT is 0 or the VM was invalid. |
| 0b10101 | A physical interrupt whose Routing mode is Targeted specified an invalid IAFFID. |
| 0b10110 | A virtual interrupt whose Routing mode is Targeted specified a VPE ID that is < 2 ^ L2_VMTE.VPE_ID_BITS but invalid otherwise. |
| 0b10111 | The IRS was trying to signal a VPE doorbell but the VPE doorbell INTID is unreachable. |
| 0b11001 | The IRS has detected corrupt metadata in the L2_ISTE. |
| 0b11010 | The IRS has detected corrupt metadata in a VPE descriptor. |
| 0b11011 | The IRS has detected corrupt metadata in a VM descriptor. |
| 0b11100 | IRS_MAP_L2_ISTR was written when IRS_IST_STATUSR.IDLE == 0. |
| 0b11101 | IRS_VMAP_L2_VMTR was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b11110 | IRS_VMAP_VMR was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b11111 | IRS_VMAP_VISTR was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b100000 | IRS_VMAP_L2_VISTR was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b100001 | IRS_VMAP_VPER was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b100010 | IRS_IST_BASER was written when IRS_IST_STATUSR.IDLE == 0. |
| 0b100011 | IRS_PE_SELR was written when IRS_PE_STATUSR.IDLE == 0. |
| 0b100100 | IRS_SAVE_VMR was written when IRS_SAVE_VM_STATUSR.IDLE == 0. |
| 0b100101 | IRS_SPI_SELR was written when IRS_SPI_STATUSR.IDLE == 0. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| EC | Meaning |
|---|---|
| 0b100110 | IRS_SYNCR was written when IRS_SYNC_STATUSR.IDLE == 0. |
| 0b100111 | IRS_VM_SELR was written when IRS_VM_STATUSR.IDLE == 0. |
| 0b101000 | IRS_VMT_BASER was written when IRS_VMT_STATUSR.IDLE == 0. |
| 0b101001 | IRS_VPE_SELR was written when IRS_VPE_STATUSR.IDLE == 0. |
| 0b101010 | The IRS has detected that L2_ISTE.IRM is 1 and 1ofN is not supported. |
| 0b101011 | A resample request for an SPI from a write to IRS_SPI_RESAMPLER failed. |

All other values are reserved.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**Bits [15:4]**

Reserved, RES0.

**OF, bit [3]**

Specifies whether multiple software errors have been detected.

When this field is 1, the syndrome information reports information about the error that last caused IRS_SWERR_STATUSR.V to transition from 0 to 1.

| OF | Meaning |
|---|---|
| 0b0 | No errors have been detected, since the error that was reported when IRS_SWERR_STATUSR.V last transitioned from 0 to 1. |
| 0b1 | At least one error has been detected, since the error that was reported when IRS_SWERR_STATUSR.V last transitioned from 0 to 1. |

When clearing IRS_SWERR_STATUSR.V to 0, if this field is nonzero, software writes 1 to clear this field to zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

**S1V, bit [2]**

Specifies whether IRS_SWERR_SYNDROMER1 is valid.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| S1V | Meaning |
|---|---|
| 0b0 | IRS_SWERR_SYNDROMER1 is not valid. |
| 0b1 | IRS_SWERR_SYNDROMER1 is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**S0V, bit [1]**

Specifies whether IRS_SWERR_SYNDROMER0 is valid.

| S0V | Meaning |
|---|---|
| 0b0 | IRS_SWERR_SYNDROMER0 is not valid. |
| 0b1 | IRS_SWERR_SYNDROMER0 is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**V, bit [0]**

Specifies whether other fields in this register are valid and at least one software error has been reported.

| V | Meaning |
|---|---|
| 0b0 | IRS_SWERR_STATUSR is not valid. |
| 0b1 | IRS_SWERR_STATUSR is valid. |

Software writes 1 to this field to clear it to zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is W1C.

### *Accessing IRS_SWERR_STATUSR*

After reading IRS_SWERR_STATUSR, software clears the valid fields in the register to allow new errors to be reported.

However, between reading the register and clearing the valid fields, a new error might have overwritten the register.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

To prevent this error being lost by software, the register prevents updates to fields that might have been updated by a new error.

This is done by ensuring a write to the register is ignored if all of the following are true:

- Any of IRS_SWERR_STATUSR.{V, OF} are nonzero before the write.
- The write does not clear the nonzero IRS_SWERR_STATUSR.{V, OF} fields to zero by writing ones to the applicable field or fields.

Accesses to this register use the following encodings:

Accessible at offset `0x03C0` from IRS_CONFIG_FRAME

- When IRS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.33 IRS_SWERR_SYNDROMER0

The IRS_SWERR_SYNDROMER0 characteristics are:

**Purpose**

IRS software error syndrome register 0. Records IRS specific software error syndrome information.

**Configuration**

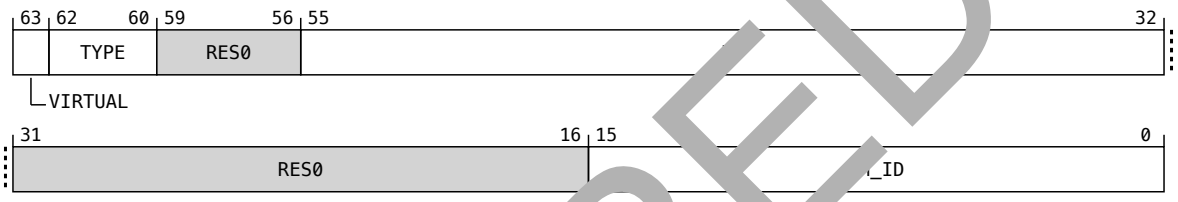
There are no configuration notes.

**Attributes**

IRS_SWERR_SYNDROMER0 is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**VIRTUAL, bit [63]**

Specifies whether the error syndrome information is for a physical or virtual interrupt.

| VIRTUAL | Meaning |
|---|---|
| 0b0 | The error syndrome information is for a physical interrupt |
| 0b1 | The error syndrome information is for a virtual interrupt |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**TYPE, bits [62:60]**

The type of the interrupt for which an error was detected.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [59:56]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**ID, bits [55:32]**

ID of the interrupt for which an error was detected.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [31:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

Bits[15:0] of the VM ID of the virtual interrupt that could not be routed by the IRS as described below.

- For a virtual LPI, this is the VM ID specified in the incoming interrupt event.
- For a virtual SPI, this is the VM ID to which the virtual SPI was assigned.

When VIRTUAL is 0, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

### *Accessing IRS_SWERR_SYNDROMER0*

Accesses to this register use the following encodings:

Accessible at offset `0x03C8` from IRS_CONFIG_FRAME.

- When IRS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- When IRS_SWERR_STATUSR.V == 1 and IRS_SWERR_STATUSR.S0V == 1, accesses to this register are RO.
- Otherwise, accesses to this register are UNKNOWN/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.34 IRS_SWERR_SYNDROMER1

The IRS_SWERR_SYNDROMER1 characteristics are:

**Purpose**

IRS software error syndrome register 1. Records IRS specific software error syndrome information.

**Configuration**

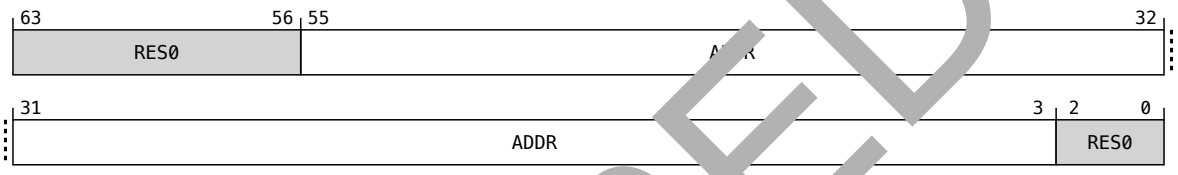
There are no configuration notes.

**Attributes**

IRS_SWERR_SYNDROMER1 is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:3]**

Bits[55:3] of the physical address of a translation structure associated with the detected error.

The address in this field is associated with the PAS of the Interrupt Domain where the error is detected.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

The reset behavior of this field is:

- On GIC reset, this field resets to an UNKNOWN value.

**Bits [2:0]**

Reserved, RES0.

***Accessing IRS_SWERR_SYNDROMER1***

Accesses to this register use the following encodings:

Accessible at offset `0x03D0` from IRS_CONFIG_FRAME

- When IRS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- When IRS_SWERR_STATUSR.V == 1 and IRS_SWERR_STATUSR.S1V == 1, accesses to this register are RO.
- Otherwise, accesses to this register are UNKNOWN/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.35 IRS_SYNCR

The IRS_SYNCR characteristics are:

**Purpose**

IRS synchronize interrupt events register.

**Configuration**

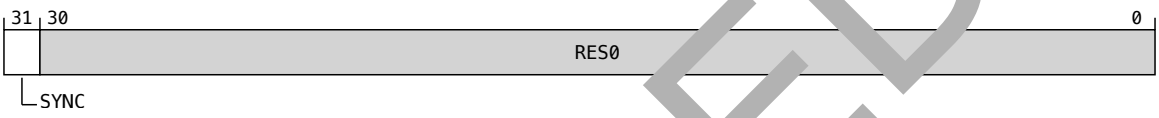
The effects of a write to this register are not guaranteed to have completed before IRS_SYNC_STATUSR.IDLE is 1.

**Attributes**

IRS_SYNCR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**SYNC, bit [31]**

Writing 1 to this field requests synchronization of interrupt events for the IRS Domain.

Writing 0 to this field has no effect.

See 'IRS synchronization requests' for more information.

| SYNC | Meaning |
|---|---|
| 0b0 | Ignored. |
| 0b1 | Issue synchronization request. |

The reset behavior of this field is:

- On CONFIG reset, this field resets to an UNKNOWN value.

**Bits [30:0]**

Reserved, RES0.

***Accessing IRS_SYNCR***

Accesses to this register use the following encodings:

Accessible at offset 0x00C0 from IRS_CONFIG_FRAME

- When IRS_SYNC_STATUSR.IDLE == 0, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.36 IRS_SYNC_STATUSR

The IRS_SYNC_STATUSR characteristics are:

**Purpose**

IRS synchronize interrupt events status register.

**Configuration**

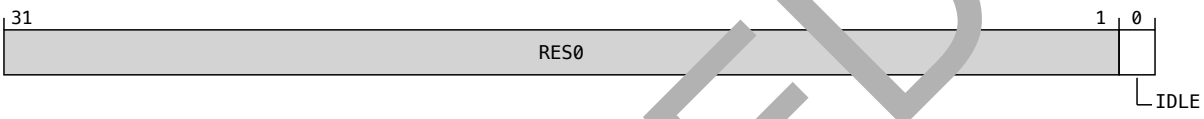
There are no configuration notes.

**Attributes**

IRS_SYNC_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Whether the effects of the last write to IRS_SYNCR have completed.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of writing to IRS_SYNCR not guaranteed to have completed. |
| 0b1 | The effects of writing to IRS_SYNCR have completed. |

The reset behavior of this field is:

- On a PIC reset, this field resets to '1'.

Access to this field is RO.

***Accessing IRS_SYNC_STATUSR***

This register is read-only.

Accesses to this register use the following encodings:

Accessible at offset 0x00C4 from IRS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.37 IRS_VM_DBR

The IRS_VM_DBR characteristics are:

**Purpose**

IRS VM 1ofN doorbell configuration register.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_VM_STATUSR.IDLE is 1.

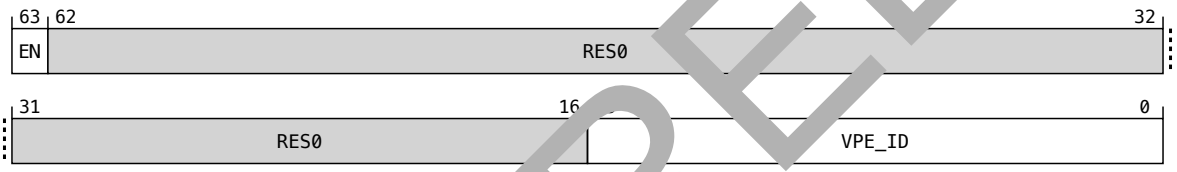
Following a write to this register, IRS_VM_STATUSR.V is updated

**Attributes**

IRS_VM_DBR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




The fields in this register return information about the VM selected in IRS_VM_SELR.

This register is updated after a write to IRS_VM_SELR when IRS_VM_STATUSR.{V, IDLE} is {1, 1}.

**EN, bit [63]**

Whether the doorbell settings are valid for the VM.

| EN | Meaning |
|---|---|
| 0b0 | 1ofN doorbells are disabled for the VM. |
| 0b1 | 1ofN doorbells are enabled for the VM. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [62:16]**

Reserved, RES0.

**VPE_ID, bits [15:0]**

The VPE ID of the 1ofN VPE doorbell target.

See ‘VPE doorbells for 1ofN interrupts’ for more information.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_VM_DBR

Accesses to this register use the following encodings:

Accessible at offset `0x0280` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0 or IRS_IDR0.VIRT_ONE_N == 0, accesses to this register are RAZ/WI.
- When [IRS_VM_STATUSR.V, IRS_VM_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.38 IRS_VM_SELR

The IRS_VM_SELR characteristics are:

**Purpose**

IRS VM selection register.

Selects a VM whose 1ofN doorbell configuration can be accessed via IRS_VM_DBR.

**Configuration**

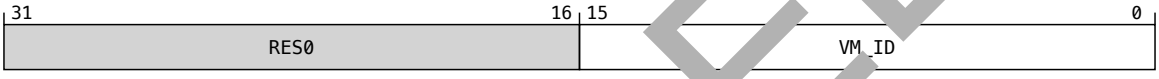
The effects of a write to this register are not guaranteed to have completed before IRS_VM_STATUSR.IDLE is 1.

**Attributes**

IRS_VM_SELR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

Identifies the VM.

***Accessing IRS_VM_SELR***

Accesses to this register use the following encodings:

Accessible at offset 0x0288 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0 or IRS_IDR0.VIRT_ONE_N == 0, accesses to this register are RAZ/WI.
- When IRS_VM_STATUSR.IDLE == 0, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.39 IRS_VM_STATUSR

The IRS_VM_STATUSR characteristics are:

**Purpose**

IRS VM status register

**Configuration**

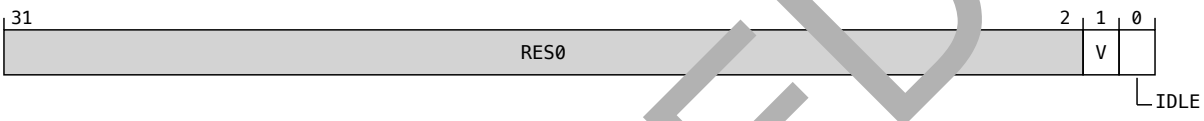
There are no configuration notes.

**Attributes**

IRS_VM_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**V, bit [1]**

Whether IRS_VM_SELR selects a valid VM.

This field is updated if and only if any of the following occur:

- A write to IRS_VM_DBR.
- A write to IRS_VM_SELR.

When IDLE is 1, the value of this field is UNKNOWN.

| V | Meaning |
|---|---|
| 0b0 | The VM selected using IRS_VM_SELR is not valid. |
| 0b1 | The VM selected using IRS_VM_SELR is valid. |

This field resets to 0 to indicate that there was no write to IRS_VM_SELR that selected a valid VM since the IRS was reset.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

**IDLE, bit [0]**

When this field returns 1 and V is 1, a read from IRS_VM_CR0 returns the VM configuration for the selected VM.

Following a write to IRS_VM_CR0, when this field returns 1 and V is 1, the effects of the write have completed:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of writing to the VM selection and configuration registers are not guaranteed to have completed. |
| 0b1 | The effects of writing to the VM selection and configuration registers have completed. |

This field resets to 1 to allow initial writes to registers Guarded by this field. Because there has been no write at reset, this does not imply that any invalidate operation is complete.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

#### *Accessing IRS_VM_STATUSR*

Accesses to this register use the following encodings:

Accessible at offset 0x028C from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0 or IRS_IDR0.VIRT_OOE_N == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.40 IRS_VMAP_L2_VISTR

The IRS_VMAP_L2_VISTR characteristics are:

**Purpose**

IRS map level 2 virtual IST register

**Configuration**

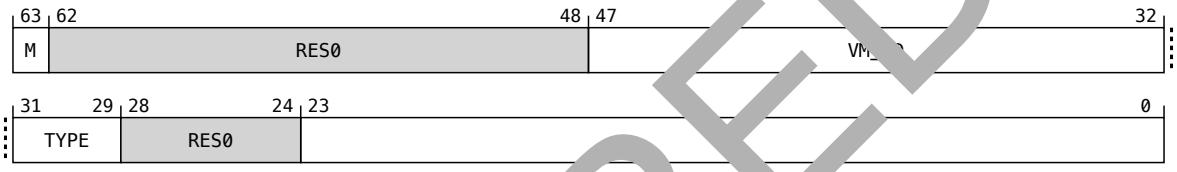
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMAP_L2_VISTR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**M, bit [63]**

Writing 1 to this field makes the virtual level 2 IST specified by TYPE and ID for the VM specified by VM_ID valid.

| M | Meaning |
|---|---|
| 0b0 | The write to this field has no effect. |
| 0b1 | The write to this field makes the specified virtual level 2 IST valid. |

There are no effects to the virtual IST on a write to this register, if any of the following are true:

- The VM specified by VM_ID is invalid.
- The virtual ID is invalid.
- The virtual IST uses a linear format.
- The INTID.ID is outside the configured virtual interrupt range for the specified VM.
- The level 2 IST is valid.
- The write to this register does not set this field to 1.

If this field is not set to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

Access to this field is WO/UNKNOWN.

**Bits [62:48]**

Reserved, RES0.

**VM_ID, bits [47:32]**

The VM ID specifying the VM for which a virtual level 2 IST is being made valid.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**TYPE, bits [31:29]**

Whether the operation applies to the virtual LPI IST or virtual SPI IST.

Values not defined are reserved.

When programming a reserved value, it is CONSTRAINED UNPREDICTABLE whether the invalidate is IGNORED or an UNKNOWN value is used for this field.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

An INTID.ID covered by the level 1 IST[illegible] corresp[illegible]ding to t[illegible]evel 2 IST that is made valid.

If unimplemented upper bits of the INT[illegible] ID [illegible] is CONSTRAINED UNPREDICTABLE whether the upper bits are treated as 0 or a[illegible]quest t[illegible]validate is IGNORED.

The reset behavior of this fie[illegible]is:

- This field resets to [illegible]KNOWN valu[illegible]

### *Accessing IRS_VMAP_L2_VISTR*

Accesses to this regi[illegible]r use the following [illegible]ncodings:

Accessible at off[illegible] 0x02D8 [illegible]om IRS_CONFIG_FRAME

- When IRS_ID[illegible] IRT == 0[illegible]ccesses to this register are RAZ/WI.
- Wh[illegible]IRS_VMT_[illegible]TATU[illegible]R.IDLE == 0, accesses to this register are RO.
- [illegible]hen IRS[illegible]VMT_B[illegible]R.VALID == 0, accesses to this register are RAZ/WI.
- Otherwi[illegible]ccesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.41 IRS_VMAP_L2_VMTR

The IRS_VMAP_L2_VMTR characteristics are:

**Purpose**

IRS map level 2 VM table register

**Configuration**

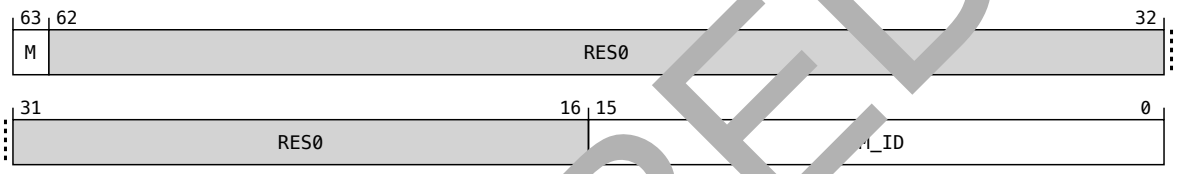
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMAP_L2_VMTR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**M, bit [63]**

Writing 1 to this field makes the level 2 VM table specified by VM_ID valid.

| M | Meaning |
|---|---|
| 0b0 | The write to this field has no effect. |
| 0b1 | The write to this field makes the specified level 2 VM table valid. |

There is no effect to the VM table on a write to this register, if any of the following are true:

- The VM table is valid.
- The VM table uses a linear structure.
- The VM ID specified by VM_ID is outside the configured VM ID range.
- The level 2 VM table is valid.
- The write to this register does not set this field to 1.

If this field is not set to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

Access to this field is WO/UNKNOWN.

**Bits [62:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

The VM ID specifying the level 2 VM table being made valid.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_VMAP_L2_VMTR

Accesses to this register use the following encodings:

Accessible at offset `0x02C0` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_CFGR.STRUCTURE == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_BASER.VALID == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.42 IRS_VMAP_VISTR

The IRS_VMAP_VISTR characteristics are:

**Purpose**

IRS map virtual IST register

**Configuration**

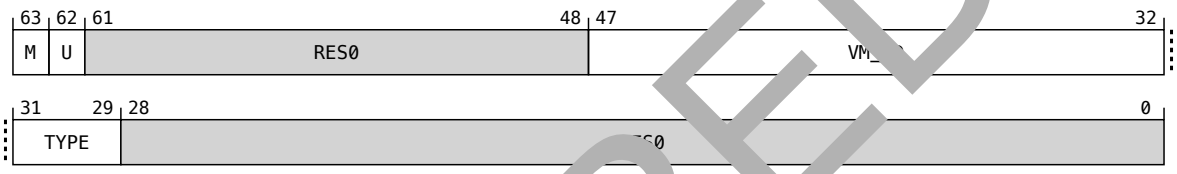
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMAP_VISTR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**M, bit [63]**

Writing 1 to this field makes the virtual IST specified by TYPE for the VM specified by VM_ID valid or invalid as specified by U.

| M | Meaning |
|---|---|
| 0b0 | The write to this field has no effect. |
| 0b1 | The write to this field makes the specified virtual IST valid. |

There are no effects on the validity of the virtual IST on a write to this register, if any of the following are true:

- The VM specified by VM_ID is invalid.
- The VM ID is outside the configured VM ID range.
- The virtual IST is valid and U is 0.
- The virtual IST is invalid and U is 1.
- The write to this register does not set this field to 1.

If this field is not set to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

Access to this field is WO/UNKNOWN.

**U, bit [62]**

Whether a write that sets M to 1 makes the specified virtual IST valid or invalid.

| U | Meaning |
|---|---|
| 0b0 | A write that sets M to 1 makes the specified virtual IST valid. |
| 0b1 | A write that sets M to 1 makes the specified virtual IST invalid. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [61:48]**

Reserved, RES0.

**VM_ID, bits [47:32]**

The VM ID specifying the VM for which a virtual IST is being made valid.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

**TYPE, bits [31:29]**

Whether the operation applies to the virtual LPI IST or virtual SPI IST.

Values not defined are reserved.

When programming a reserved value, it is CONSTRAINED UNPREDICTABLE whether the invalidate is IGNORED or an UNKNOWN value is used for this field.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

**Bits [28:0]**

Reserved, RES0.

### *Accessing IRS_VMAP_VISTR*

Accessible at offset 0x02F8 from IRS_CONFIG_FRAME

- When IRS_IDR6.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- When IRS_VMT_BASER.VALID == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.43 IRS_VMAP_VMR

The IRS_VMAP_VMR characteristics are:

**Purpose**

IRS map VM register

**Configuration**

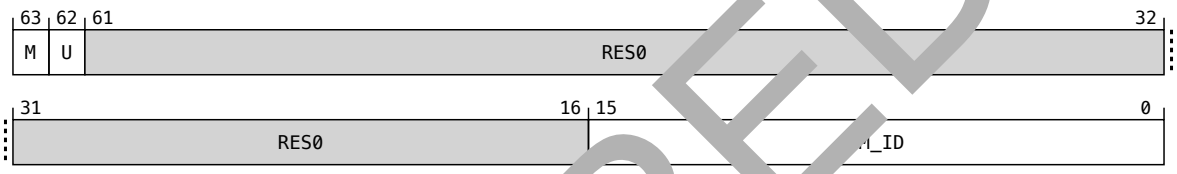
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMAP_VMR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**M, bit [63]**

Writing 1 to this field makes the VM specified by VM_ID valid or invalid as specified by U.

| M | Meaning |
|---|---|
| 0b0 | The write to this field has no effect. |
| 0b1 | The write to this field makes the specified VM valid or invalid. |

There is no effect to the validity of the VM on a write to this register, if any of the following are true:

- The level 2 VMAP page containing the entry corresponding to the VM specified by VM_ID is invalid.
- The VM_ID is outside the configured VM ID range.
- The VM valid and U is 0.
- The VM invalid and U is 1.
- The write to this register does not set this field to 1.

If this field is not set to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

Access to this field is WO/UNKNOWN.

**U, bit [62]**

Whether a write that sets M to 1 makes the specified VM valid or invalid.

| U | Meaning |
|---|---|
| 0b0 | A write that sets M to 1 makes the specified VM valid. |
| 0b1 | A write that sets M to 1 makes the specified VM invalid. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [61:16]**

Reserved, RES0.

**VM_ID, bits [15:0]**

The VM ID specifying the VM being made valid or invalid.

#### *Accessing IRS_VMAP_VMR*

Accesses to this register use the following encodings:

Accessible at offset `0x02C8` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_BASER.VALID == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.44 IRS_VMAP_VPER

The IRS_VMAP_VPER characteristics are:

**Purpose**

IRS map VPE register

**Configuration**

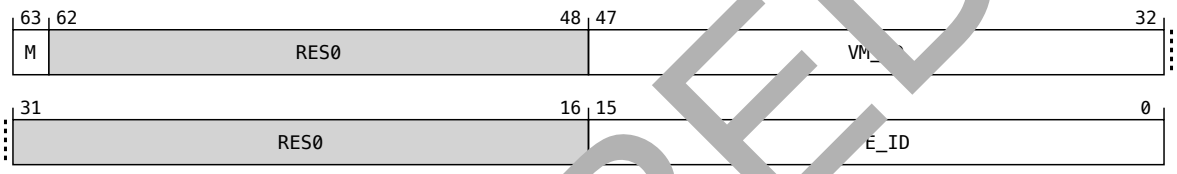
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMAP_VPER is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**M, bit [63]**

Writing 1 to this field makes the V[illegible] specified by VM_ID and VPE_ID valid.

| M | Meaning |
|---|---|
| 0b0 | The write to this field has no effect. |
| 0b1 | The write to this field makes the specified VPE valid. |

There is no effect to the validity of the VPE on a write to this register, if any of the following are true:

- The VM specified by VM_ID is invalid.
- The VM ID is outside the configured VM ID range.
- The VPE ID is outside the configured VPE range for the VM.
- The VPE is valid.
- This write to this register does not set this field to 1.

If this field is not set to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

Access to this field is WO/UNKNOWN.

**Bits [62:48]**

Reserved, RES0.

**VM_ID, bits [47:32]**

Identifies the VM.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [31:16]**

Reserved, RES0.

**VPE_ID, bits [15:0]**

Identifies the VPE.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

#### *Accessing IRS_VMAP_VPER*

Accesses to this register use the following encodings:

Accessible at offset 0x02E0 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- When IRS_VMT_BASER.VALID == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.45 IRS_VMT_BASER

The IRS_VMT_BASER characteristics are:

**Purpose**

IRS VM table base address register

**Configuration**

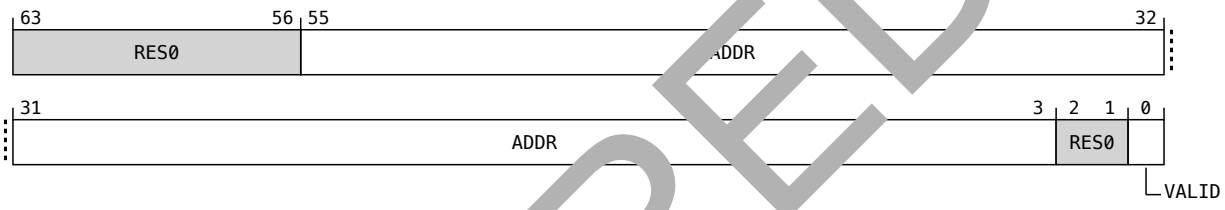
The effects of a write to this register are not guaranteed to have completed before IRS_VMT_STATUSR.IDLE is 1.

**Attributes**

IRS_VMT_BASER is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:3]**

Bits[55:3] of the VM table base physical address.

This field is Guarded by IRS_VMT_BASER.VALID.

When IRS_VMT_CFGR.STRUCTURE is 0, ADDR points to a linear VM table and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N = 4 + IRS_VMT_CFGR.VM_ID_BITS.
- This means that the level 2 VMTE array is aligned to the size of the array.

When IRS_VMT_CFGR.STRUCTURE is 1, ADDR points to the level 1 table in a 2-level VM table and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N depends on VM_ID_BITS in IRS_VMT_CFGR as follows:

  N = Max(2, VM_ID_BITS - 7 + 2)

- This means that the level 1 VMT is aligned to the size of the level 1 VMTE array.

Access to any level of the VM table and any additional memory accesses occurring as a result of the address in this field are performed using the PAS of the IRS Domain where this register is accessed.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When IRS_VMT_BASER.VALID == 1, access to this field is RO.
- Otherwise, access to this field is RW.

**Bits [2:1]**

Reserved, RES0.

**VALID, bit [0]**

Whether the ADDR points to a valid VM table.

| VALID | Meaning |
|---|---|
| 0b0 | The VM table address is not valid. |
| 0b1 | The VM table address is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

### *Accessing IRS_VMT_BASER*

Accesses to this register use the following encodings:

Accessible at offset 0x0200 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.46 IRS_VMT_CFGR

The IRS_VMT_CFGR characteristics are:

**Purpose**

IRS VM table configuration register

**Configuration**

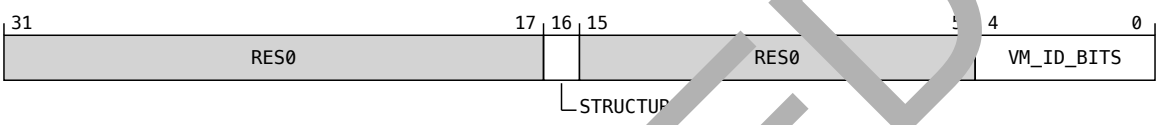
There are no configuration notes.

**Attributes**

IRS_VMT_CFGR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:17]**

Reserved, RES0.

**STRUCTURE, bit [16]**

Whether the VM table uses a linea[illegible]r 2-lev[illegible]tructure.

| STR[illegible]CTUR[illegible] | Meaning |
|---|---|
| 0b0 | A linear VM table structure is used. |
| 0b1 | A 2-level VM table structure is used. |

I[illegible]VM_ID[illegible]BITS < [illegible]STRUCTURE is 1, all of the following are true:

- Th[illegible]ists of a single level 1 VM table entry and a single level 2 VM table.
- [illegible]e level 2 VM table contains (2 ^ VM_ID_BITS) entries.
- [illegible]e IRS is allowed to access the full 4KB level 2 VM table.

Arm reco[illegible]mends that STRUCTURE is 0 when VM_ID_BITS < 8.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [15:5]**

Reserved, RES0.

**VM_ID_BITS, bits [4:0]**

The number of VM_ID bits for the IRS Domain.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than a direct read of the register. The maximum value is reported in IRS_IDR3.VM_ID_BITS.

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an UNKNOWN value.

#### Accessing IRS_VMT_CFGR

Accesses to this register use the following encodings:

Accessible at offset `0x0210` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VMT_BASER.Valid == 1 or IRS_VMT_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.47 IRS_VMT_STATUSR

The IRS_VMT_STATUSR characteristics are:

**Purpose**

IRS virtualization data structures management status register.

**Configuration**

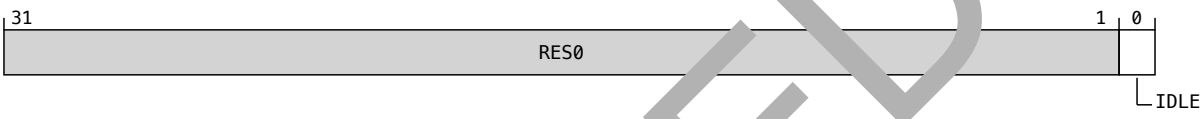
There are no configuration notes.

**Attributes**

IRS_VMT_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Whether the effects of any of the following writes are complete:

- A write that updates IRS_VMT_BASER.Valid
- A write that sets IRS_VMAP_L2_VMTR.M to 1.
- A write that sets IRS_VMAP_VMR.M to 1.
- A write that sets IRS_VMAP_VPER.M to 1.
- A write that sets IRS_VMAP_VISTR.M to 1.
- A write that sets IRS_VMAP_L2_VISTR.M to 1.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects a write to any of the virtualization data structure management registers are not complete |
| 0b1 | The effects a write to any of the virtualization data structure management registers are complete |

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Access to this field is RO.

***Accessing IRS_VMT_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x0214 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.48 IRS_VPE_CR0

The IRS_VPE_CR0 characteristics are:

**Purpose**

IRS VPE Control Register 0

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_VPE_STATUSR.IDLE is 1.

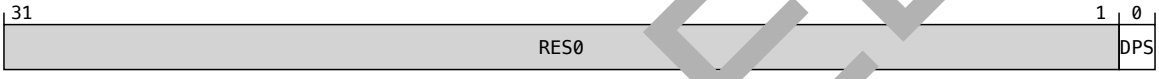
Following a write to this register, IRS_VPE_STATUSR.V is updated.

**Attributes**

IRS_VPE_CR0 is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register is updated after a write that sets IRS_VPE_SELR.S to 0 when IRS_VPE_STATUSR.{V, IDLE} is {1, 1}.

**Bits [31:1]**

Reserved, RES0.

**DPS, bit [0]**

**When IRS_IDR0.VIRT_ONE_N == 1:**

Disable 1ofN PE selection.

| DPS | Meaning |
|---|---|
| 0b0 | 1ofN PE selection is enabled. A virtual interrupt configured to use the 1ofN Routing mode is permitted to select this VPE. |
| 0b1 | 1ofN PE selection is disabled. A virtual interrupt configured to use the 1ofN Routing mode is not permitted to select this VPE. |

This field determines whether the selected VPE is permitted to be selected for 1ofN interrupt delivery.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IRS_VPE_CR0

Accesses to this register use the following encodings:

Accessible at offset `0x0258` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When [IRS_VPE_STATUSR.V, IRS_VPE_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.49 IRS_VPE_DBR

The IRS_VPE_DBR characteristics are:

**Purpose**

IRS VPE doorbell settings register.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before IRS_VPE_STATUSR.IDLE is 1.

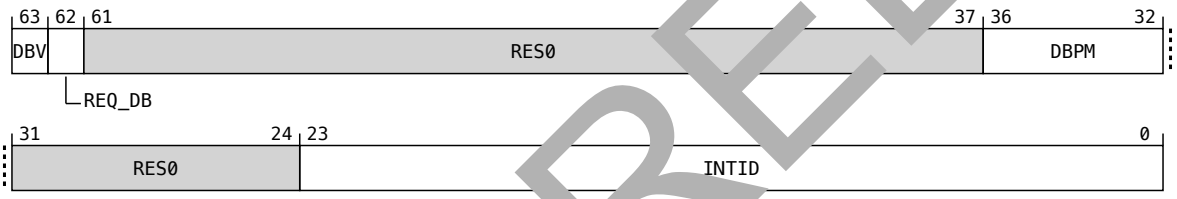
Following a write to this register, IRS_VPE_STATUSR.V is updated.

**Attributes**

IRS_VPE_DBR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register is updated after a write that sets IRS_VPE_SELR.S to 1 when IRS_VPE_STATUSR.{V, IDLE} is {1, 1}.

**DBV, bit [63]**

Whether the doorbell settings for the VPE are valid.

| DBV | Meaning |
|---|---|
| 0b0 | The doorbell settings for the VPE are not valid. |
| 0b1 | The doorbell settings for the VPE are valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**REQ_DB, bit [62]**

Whether a doorbell is requested for the VPE.

When a VPE doorbell is requested for the VPE, the doorbell event is generated when the VPE doorbell conditions are met.

See ‘VPE doorbells’ for more information.

| REQ_DB | Meaning |
|---|---|
| 0b0 | A doorbell is not requested for the VPE |
| 0b1 | A doorbell is requested for the VPE. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_VPE_DBR.DBV == 1, access to this field is RW.
- Otherwise, access to this field is WO/UNKNOWN.

**Bits [61:37]**

Reserved, RES0.

**DBPM, bits [36:32]**

Doorbell priority mask.

This field specifies the minimum priority for a virtual interrupt to trigger the VPE's doorbell.

Accessing this field has the following behavior:

- When IRS_VPE_DBR.REQ_DB == 1, access to this field is RW.
- Otherwise, access to this field is WO/UNKNOWN.

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

If IRS_IDR2.LPI is 1, this field specifies the LPI INTID ID of the VPE doorbell.

The number of ID bits implemented is reported in IRS_IDR2.ID_BITS. Unimplemented upper bits are RES0.

If IRS_IDR2.LPI is 0, this field is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IRS_VPE_DBR.DBV == 1, access to this field is RW.
- Otherwise, access to this field is WO/UNKNOWN.

#### *Accessing IRS_VPE_DBR*

Accesses to this register use the following encodings:

Accessible at offset `0x0248` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When [IRS_VPE_STATUSR.V, IRS_VPE_STATUSR.IDLE] != 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.50 IRS_VPE_HPPIR

The IRS_VPE_HPPIR characteristics are:

**Purpose**

IRS VPE HPPI register.

**Configuration**

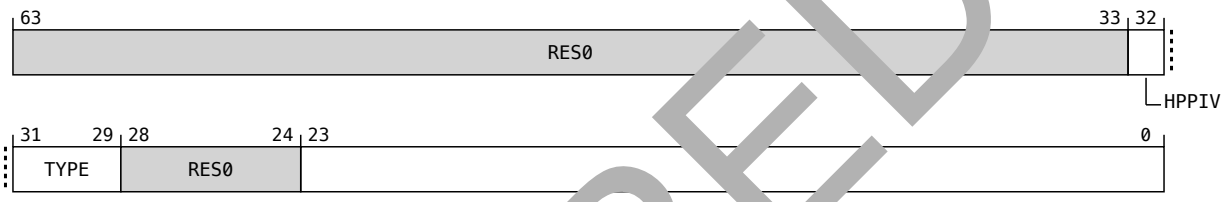
There are no configuration notes.

**Attributes**

IRS_VPE_HPPIR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




This register reports information about the candidate HPPI for the VPE selected by IRS_VPE_SELR.

This register is updated after a write that sets IRS_VPE_SELR.S to 1 when IRS_VPE_STATUSR.{V, IDLE} is {1, 1}.

If there is a change in the HPPI for the VPE following the write that set IRS_VPE_SELR.S to 1, it is IMPLEMENTATION DEFINED whether the old or the new HPPI is reported in this register.

**Bits [63:33]**

Reserved, RES0.

**HPPIV, bit [32]**

When 1, there is a candidate HPPI for the VPE.

| HPPIV | Meaning |
|---|---|
| 0b0 | Invalid: There is no candidate HPPI for the VPE. |
| 0b1 | VALID: There is a candidate HPPI for the VPE. |

When the value of this field is 1, ID and TYPE together form the INTID of the candidate HPPI for the VPE.

**TYPE, bits [31:29]**

The encoding of this field depends upon the value in HPPIV as described below:

- If HPPIV is 0, this field is RES0.
- If HPPIV is 1, this field contains valid information.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| TYPE | Meaning |
|---|---|
| 0b011 | SPI |

Values not defined above are reserved.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

The encoding of this field depends upon the value in HPPIV as described below:

- If HPPIV is 0, this field is RES0.
- If HPPIV is 1, this field contains valid information.

#### *Accessing IRS_VPE_HPPIR*

Accesses to this register use the following encodings:

Accessible at offset 0x0250 from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI
- When [IRS_VPE_STATUSR.V, IRS_VPE_STATUSR.IDAM] = 0b11, accesses to this register are UNKNOWN/WI.
- Otherwise, accesses to this register are

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.51 IRS_VPE_SELR

The IRS_VPE_SELR characteristics are:

**Purpose**

IRS VPE selection register.

**Configuration**

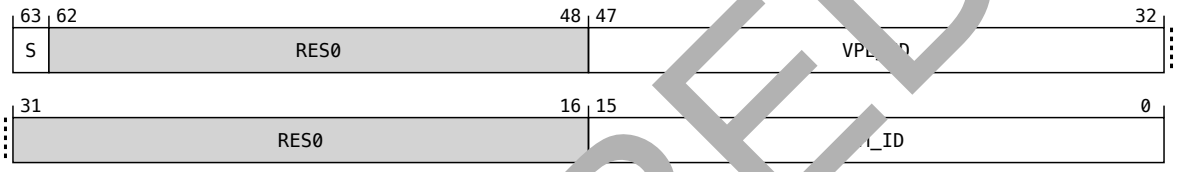
The effects of a write to this register are not guaranteed to have completed before IRS_VPE_STATUSR.IDLE is 1.

**Attributes**

IRS_VPE_SELR is a 64-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**S, bit [63]**

Writing 1 to this field selects a VPE whose configuration can be accessed via the following registers:

- IRS_VPE_DBR
- IRS_VPE_HPPIR
- IRS_VPE_CR0

| S | Meaning |
|---|---|
| 0b0 | The write to this register has no effect. |
| 0b1 | The write to this register selects a VPE. |

This field is RES1 to 1 on a write to this register, updates to any other field in this register has no effect beyond updating the value of that field.

This means that if IRS_VPE_STATUSR.V is updated as a result of a write to another register, the value returned reflects the VPE selected when this field was set to 1.

Access to this field is WO/UNKNOWN.

**Bits [62:48]**

Reserved, RES0.

**VPE_ID, bits [47:32]**

Identifies the VPE.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

**Bits [31:16]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**VM_ID, bits [15:0]**

Identifies the VM.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

### *Accessing IRS_VPE_SELR*

Accesses to this register use the following encodings:

Accessible at offset `0x0240` from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- When IRS_VPE_STATUSR.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.1.52 IRS_VPE_STATUSR

The IRS_VPE_STATUSR characteristics are:

**Purpose**

IRS VPE status register

**Configuration**

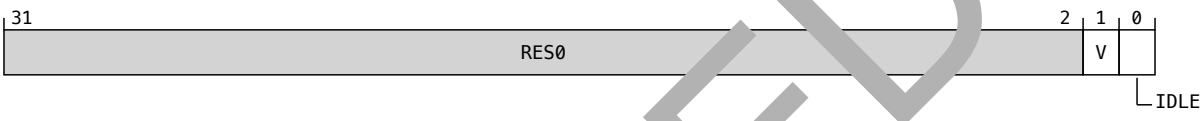
There are no configuration notes.

**Attributes**

IRS_VPE_STATUSR is a 32-bit register.

This register is part of the IRS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**V, bit [1]**

Whether the value last written to IRS_VPE_SELR selects a valid VPE.

This field is updated if and only if any of the following occur:

- A write to IRS_VPE_CR0.
- A write to IRS_VPE_DBR.
- A write that sets IRS_VPE_SELR.S to 1.

When IDLE is 0, the value of this field is UNKNOWN.

| | Meaning |
|---|---|
| 0b0 | The VPE selected using IRS_VPE_SELR is not valid. |
| 0b1 | The VPE selected using IRS_VPE_SELR is valid. |

This field resets to 0 to indicate that there was no write to IRS_VPE_SELR that selected a valid VPE that is managed on this IRS since the IRS was reset.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

**IDLE, bit [0]**

A write that sets IRS_VPE_SELR.S to 1 is complete when this field is 1.

When this field is 1 and V is 1, a read from any of the following registers returns values for the selected VPE:

- IRS_VPE_DBR.
- IRS_VPE_HPPIR.
- IRS_VPE_CR0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Following a write to one of the following registers, when this field returns 1, the effects of the write have completed:

- IRS_VPE_DBR.
- IRS_VPE_CR0.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the last write that set IRS_VPE_SELR.S to 1 and any write to VPE configuration registers are not guaranteed to be complete. |
| 0b1 | The effects of the last write that set IRS_VPE_SELR.S to 1 and any write to VPE configuration registers are complete. |

This field resets to 1 to allow initial writes to registers Guarded by this field. Because there has been no write at reset, this does not imply that any invalidate operation is complete.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

#### *Accessing IRS_VPE_STATUSR*

Accesses to this register use the following encodings:

Accessible at offset 0x025C from IRS_CONFIG_FRAME

- When IRS_IDR0.VIRT == 0, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

### 10.2.2 IRS_SETLPI_FRAME

The IRS_SETLPI_FRAME characteristics are:

#### Purpose

Contains the IRS_SETLPI register used to generate a SET_EDGE message for an LPI without going through an ITS.

An IRS_SETLPI register frame is present for each supported Interrupt Domain on each IRS where IRS_IDR0.SETLPI is 1.

Arm strongly recommends that this register frame is not accessible by PEs. This is because a write to this register can require the IRS access to memory, leading to in-out dependencies than can potentially lead to deadlocks in the system. If this register frame is not accessible by PEs, the behavior on an attempted access from a PE is IMPLEMENTATION DEFINED and is likely to result in an External abort.

This register frame is only accessible in the PAS associated with the Interrupt Domain.

The base address is distinct from addresses of registers accessible in any other PAS.

The base address is aligned to 64KB.

#### Configuration

This Register Block is present only when IRS_IDR0.SETLPI == 1

#### Attributes

IRS_SETLPI_FRAME is a block of size: 0x10000 bytes

#### Contents

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| Offset | Name |
|---|---|
| 0x0000 | IRS_SETLPIR |

Direct accesses to other offsets in this block are RAZ/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.2.2.1 IRS_SETLPIR

The IRS_SETLPIR characteristics are:

**Purpose**

IRS SETLPI register.

A write to this register generates a SET_EDGE message for the LPI ID specified as part of the write.

**Configuration**

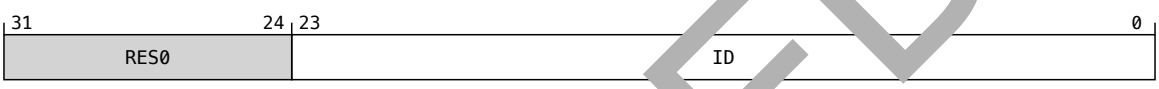
There are no configuration notes.

**Attributes**

IRS_SETLPIR is a 32-bit register.

This register is part of the IRS_SETLPI_FRAME block.

***Field descriptions***




**Bits [31:24]**

Reserved, RES0.

**ID, bits [23:0]**

Bits[23:0] of the LPI INTID.ID to se[illegible]enerate[illegible] SET_EDGE message for.

If unimplemented upper bits of the INTID.ID are not zero, it is IMPLEMENTATION DEFINED whether the upper bits are treated as 0 or the interrupt message is ignored by the IRS.

***Accessing IRS_SETLPIR***

Writes to this register are ignored if the IRS is not enabled for the Interrupt Domain.

Accesses to this register use the following encodings:

Accessible at offset 0x0000 from IRS_SETLPI_FRAME

Accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 10.3 ITS register frames

$I_{DQKQD}$ For each Interrupt Domain, the ITS exposes the following register frames:

- The ITS configuration register frame.
- The ITS translation register frames.

Different from GICv3, the GICv5 architecture does not require the ITS register frames to be contiguous with respect to each other.

$I_{BHGQG}$ In the ITS register definitions, references to other registers are always to a register in the register frame for the same ITS Domain as where the access is made.

### 10.3.1 ITS_CONFIG_FRAME

The ITS_CONFIG_FRAME characteristics are:

#### Purpose

Contains control registers for an ITS Domain.

An ITS configuration register frame is present for each supported ITS Domain.

This register frame is accessible in the PAS associated with the ITS Domain.

It is IMPLEMENTATION DEFINED whether this register frame is also accessible in the MPPAS at the same address.

The base address is distinct from the base address of any other GIC register frame, including the ITS configuration register frames for other ITS Domains.

The base address is aligned to 64KB.

#### Attributes

ITS_CONFIG_FRAME is a block of size: 0x10000 bytes

#### Contents

| Offset | Name |
|---|---|
| 0x0000 | ITS_IDR0 |
| 0x0004 | ITS_IDR1 |
| 0x0008 | ITS_IDR2 |
| 0x0040 | ITS_IIDR |
| 0x0044 | ITS_AIDR |
| 0x0080 | ITS_CR0 |
| 0x0084 | ITS_CR1 |
| 0x00C0 | ITS_DT_BASER |
| 0x00D0 | ITS_DT_CFGR |
| 0x0100 | ITS_DIDR |
| 0x0108 | ITS_EIDR |
| 0x010C | ITS_INV_EVENTR |
| 0x0110 | ITS_INV_DEVICER |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Offset | Name |
|---|---|
| 0x0114 | ITS_READ_EVENTR |
| 0x0118 | ITS_READ_EVENT_DATAR |
| 0x0120 | ITS_STATUSR |
| 0x0140 | ITS_SYNCR |
| 0x0148 | ITS_SYNC_STATUSR |
| 0x0180 | ITS_GEN_EVENT_DIDR |
| 0x0188 | ITS_GEN_EVENT_EIDR |
| 0x018C | I[illegible]EVENTR |
| 0x0190 | ITS_GEN_[illegible]ENT_STATUSR |
| 0x01C[illegible] | [illegible]S_MEC_I[illegible]R |
| 0[illegible]1C4 | ITS_[illegible]MECID_R |
| [illegible]0200 | ITS_MPAM_IDR |
| 0x02[illegible] | I[illegible]S_MPAM_PARTID_R |
| 0x0240 | ITS_SWERR_STATUSR |
| [illegible] | ITS_SWERR_SYNDROMER0 |
| 0x0250 | ITS_SWERR_SYNDROMER1 |
| 0XE00 - ↪0XEFC | IMPLEMENTATION DEFINED |

Direct accesses to o[illegible]r offsets in this blo[illegible]are RAZ/WI.

**Note**

Fo[illegible]nformat[illegible]n on the u[illegible]of offsets 0xE00 - 0xEFC see 10.6 *CoreSight identification registers*.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 10.3.1.1 ITS_AIDR

The ITS_AIDR characteristics are:

**Purpose**

ITS Architecture Identification Register. Identifies the GIC architecture version to which the implementation conforms.

**Configuration**

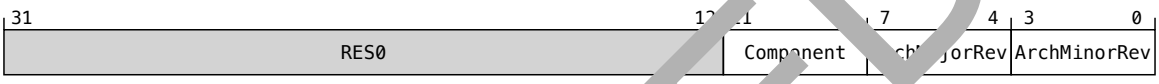
There are no configuration notes.

**Attributes**

ITS_AIDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:12]**

Reserved, RES0.

**Component, bits [11:8]**

GIC component

| Component | Meaning |
|---|---|
| 0b0000 | IRS |
| 0b0001 | ITS |
| 0b0010 | IWB |

**ArchMajorRev, bits [7:4]**

Major Architecture revision.

| ArchMajorRev | Meaning |
|---|---|
| 0b0000 | GICv5.x |

**ArchMinorRev, bits [3:0]**

Minor Architecture revision.

| ArchMinorRev | Meaning |
|---|---|
| 0b0000 | GICv5.0 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing ITS_AIDR

Accesses to this register use the following encodings:

Accessible at offset `0x0044` from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.2 ITS_CR0

The ITS_CR0 characteristics are:

**Purpose**

ITS configuration register 0

**Configuration**

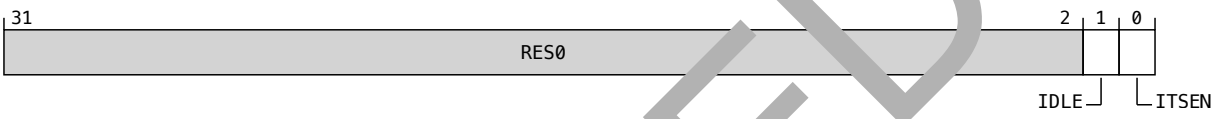
There are no configuration notes.

**Attributes**

ITS_CR0 is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**IDLE, bit [1]**

Whether the transition between enabled and disabled states of the ITS Domain is complete.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of updating ITSEN are not guaranteed to have completed. |
| 0b1 | The effects of updating ITSEN have completed. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Access to this field is RO.

**ITSEN, bit [0]**

Controls if the ITS Domain is enabled and whether it can generate interrupt messages to an IRS for the Interrupt Domain.

This field is Guarded by ITS_CR0.IDLE.

| ITSEN | Meaning |
|---|---|
| 0b0 | Disabled. The ITS does not generate any interrupt messages for the Interrupt Domain |
| 0b1 | Enabled. The ITS may generate interrupt messages for the Interrupt Domain |

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Accessing this field has the following behavior:

- When ITS_CR0.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

#### *Accessing ITS_CR0*

Accesses to this register use the following encodings:

Accessible at offset `0x0080` from ITS_CONFIG_FRAME

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.3 ITS_CR1

The ITS_CR1 characteristics are:

**Purpose**

ITS configuration register 1

**Configuration**

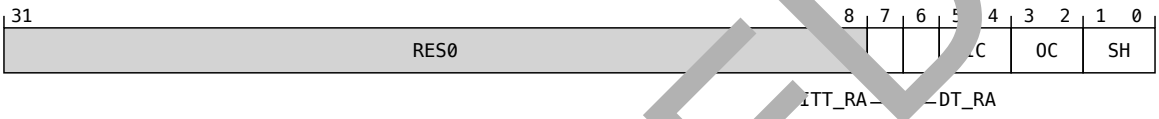
There are no configuration notes.

**Attributes**

ITS_CR1 is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:8]**

Reserved, RES0.

**ITT_RA, bit [7]**

Read-Allocate hint for the interrupt translation tables.

| ITT_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**DT_RA, bit [6]**

Read-Allocate hint for the device table.

| DT_RA | Meaning |
|---|---|
| 0b0 | No Read-Allocate. |
| 0b1 | Read-Allocate. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**IC, bits [5:4]**

Controls the Inner Cacheability attribute used when the ITS accesses tables as a requester.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| IC | Meaning |
|---|---|
| 0b00 | Non-cacheable. |
| 0b01 | Write-Back Cacheable. |
| 0b10 | Write-Through Cacheable. |
| 0b11 | Reserved, treated as 0b00. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**OC, bits [3:2]**

Controls the Outer Cacheability attribute used when the ITS accesses tables as a requester.

| OC | Meaning |
|---|---|
| 0b00 | Non-cacheable |
| 0b01 | Write-Back Cacheable. |
| 0b10 | Write-Through Cacheable. |
| 0b11 | Reserved, treated as 0b00. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**SH, bits [1:0]**

Controls the Shareability attribute used when the ITS accesses tables as a requester.

| SH | Meaning |
|---|---|
| 0b00 | Non-shareable. |
| 0b01 | Reserved, treated as 0b00. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

When ITS_CR1.OC is 0b00 and ITS_CR1.IC is 0b00, this field is IGNORED and behaves as Outer Shareable.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

### *Accessing ITS_CR1*

Accesses to this register use the following encodings:

Accessible at offset 0x0084 from ITS_CONFIG_FRAME

- When ITS_CR0.ITSEN == 1 or ITS_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.3.1.4 ITS_DIDR

The ITS_DIDR characteristics are:

**Purpose**

ITS DeviceID Register.

This register is used to specify the DeviceID for the following requests:

- Invalidation of cached information for DeviceIDs issued by a write to ITS_INV_DEVICER.
- Invalidation of cached information for EventIDs issued by a write to ITS_INV_EVENTR.
- Requesting the read of the translation for an event by a write to ITS_READ_EVENTR.

**Configuration**

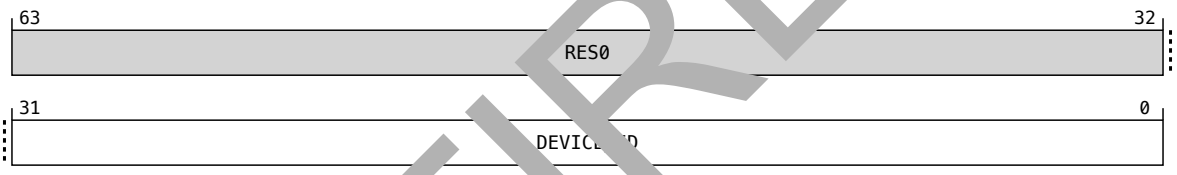
There are no configuration notes.

**Attributes**

ITS_DIDR is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:32]**

Reserved, RES0

**DEVICE_ID, bits [31:0]**

The DeviceID this should apply to a cache invalidation or reading an event translation request.

The reset behavior of this field is:

- On a PIC reset, this field resets to an UNKNOWN value.

***Accessing ITS_DIDR***

Accesses to this register use the following encodings:

Accessible at offset `0x0100` from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_STATUSR.IDLE != 1, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.5 ITS_DT_BASER

The ITS_DT_BASER characteristics are:

**Purpose**

ITS device table base address register

**Configuration**

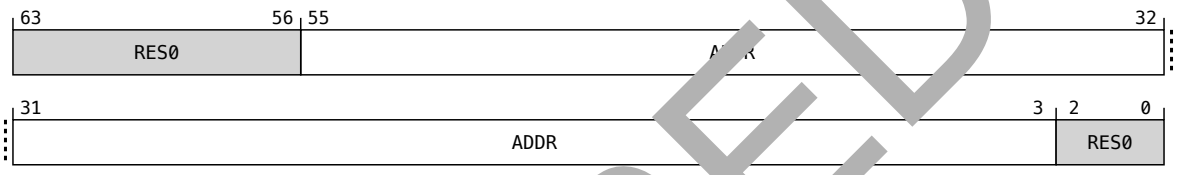
There are no configuration notes.

**Attributes**

ITS_DT_BASER is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:3]**

Bits[55:3] of the DT base physical address.

When ITS_DT_CFGR.STRUCTURE is 0, ADDR points to a linear DT and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N = 2 + ITS_DT_CFGR.DEVICEID_BITS.
- This means that the level 2 DTE array is aligned to the size of the array.

When ITS_DT_CFGR.STRUCTURE is 1, ADDR points to the level 1 table in a 2-level DT and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N depends on L2SZ and DEVICEID_BITS in ITS_DT_CFGR as follows:

  max(2, DEVICEID_BITS - (9 + (2 * L2SZ)) + 2)

- This means that the level 1 DT is aligned to the size of the table.

Access to any level of the DT and any additional memory accesses occurring as a result of the address in this field are performed using the PAS of the ITS Domain where this register is accessed.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in ITS_IDR0.PA_RANGE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [2:0]**

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing ITS_DT_BASER

Accesses to this register use the following encodings:

Accessible at offset `0x00C0` from ITS_CONFIG_FRAME

- When ITS_CR0.ITSEN == 1 or ITS_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.6 ITS_DT_CFGR

The ITS_DT_CFGR characteristics are:

**Purpose**

ITS device table base address configuration register

**Configuration**

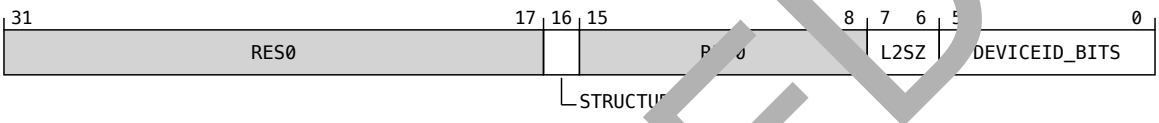
There are no configuration notes.

**Attributes**

ITS_DT_CFGR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:17]**

Reserved, RES0.

**STRUCTURE, bit [16]**

Whether the device table uses a linear or 2-level structure.

If ITS_IDR1.DT_LEVELS == 0, this field is RES0.

| STRUCTURE | Meaning |
|---|---|
| 0b0 | A linear device table structure is used. |
| 0b1 | A 2-level device table structure is used. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [15:8]**

Reserved, RES0.

**L2SZ, bits [7:6]**

Level 2 device table size when a 2-level device table structure is used.

| L2SZ | Meaning |
|---|---|
| 0b00 | A level 2 device table is maximum 4KB and resolves 9 bits of DeviceID. |
| 0b01 | A level 2 device table is maximum 16KB and resolves 11 bits of DeviceID. |
| 0b10 | A level 2 device table is maximum 64KB and resolves 13 bits of DeviceID. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Values not defined above are reserved.

If DEVICEID_BITS <= (9 + (2 * L2SZ)) and STRUCTURE is 1, the DT consists of a single L1_DTE and a single L2_DT.

The L2_DT contains (2 ^ DEVICEID_BITS) entries.

When the value of L1_DTE.SPAN is programmed to a value larger than DEVICEID_BITS, the ITS is allowed to access memory in the range specified by L1_DTE.SPAN.

Arm recommends that STRUCTURE is 0 when DEVICEID_BITS <= (9 + (2 * L2SZ)).

When programming a reserved value or an unsupported value, the ITS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

When STRUCTURE is 0, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**DEVICEID_BITS, bits [5:0]**

The number of DeviceID bits which can be translated for the accessing Security state by the ITS.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than a direct read of this register. The maximum value is reported in ITS_IDR1.DEVICEID_BITS.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

### *Accessing ITS_DT_CFGR*

Accesses to this register use the following encodings:

Accessible at offset 0x00D0 from ITS_CONFIG_FRAME

- When ITS_CR0.ITSEN == 1 or ITS_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.7 ITS_EIDR

The ITS_EIDR characteristics are:

**Purpose**

ITS EventID Register.

This register is used to specify the EventID for the following requests:

- Invalidation of cached information for EventIDs issued by a write to ITS_INV_EVENTR.
- Requesting the read of the configuration for an event by a write to ITS_READ_EVENTR.

**Configuration**

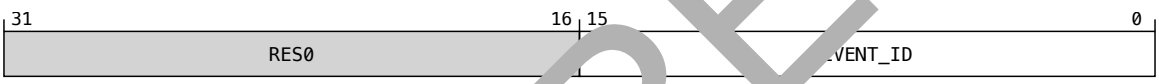
There are no configuration notes.

**Attributes**

ITS_EIDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**EVENT_ID, bits [15:0]**

The EventID as it should apply to a cache invalidation or reading an event translation request.

The reset behaviour of this field is:

- This field resets to an UNKNOWN value.

***Accessing ITS_EIDR***

Accesses to this register use the following encodings:

Accessible at offset 0x0108 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_STATUSR.IDLE != 1, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.8 ITS_GEN_EVENTR

The ITS_GEN_EVENTR characteristics are:

**Purpose**

ITS generate incoming event register.

This register is used to generate a SET_EDGE event for the EventID specified in ITS_GEN_EVENT_EIDR and the DeviceID specified in ITS_GEN_EVENT_DIDR.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before ITS_GEN_EVENT_STATUSR.IDLE is 1.

**Attributes**

ITS_GEN_EVENTR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**R, bit [31]**

Request the ITS to generate an incoming event.

| R | Meaning |
|---|---|
| 0b0 | The write has no effect on the ITS. |
| 0b1 | Generate an incoming event with the following information:<br>• The event is associated with the Interrupt Domain associated with the PAS of the write.<br>• The event is translated in the ITS Domain specified in TARGET_DOMAIN.<br>• DeviceID is specified in the ITS_GEN_EVENT_DIDR register.<br>• EventID is specified in the ITS_GEN_EVENT_EIDR register. |

**Bits [30:2]**

Reserved, RES0.

**TARGET_DOMAIN, bits [1:0]**

Specifies the ITS Domain in which the incoming event is translated.

If the specified ITS Domain is not enabled, the write to this register does not generate an incoming event.

| TARGET_DOMAIN | Meaning | Applies when |
|---|---|---|
| 0b00 | The incoming event is translated in the ITS Domain specified by ITS_IDR0.INT_DOM. | |
| 0b01 | The incoming event is translated in the Realm ITS Domain. | ITS_IDR0.INT_DOM == 0b01 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Values not defined above are reserved.

Values corresponding to unimplemented Domains are reserved.

If a reserved value is programmed, the ITS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid value.

#### *Accessing ITS_GEN_EVENTR*

Accesses to this register use the following encodings:

Accessible at offset `0x018C` from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_GEN_EVENT_STATUSR.IDLE != 1, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.9 ITS_GEN_EVENT_STATUSR

The ITS_GEN_EVENT_STATUSR characteristics are:

**Purpose**

ITS generate incoming event status register.

Reports the status of a request to generate an incoming event via a write to the ITS_GEN_EVENTR register.

**Configuration**

There are no configuration notes.

**Attributes**

ITS_GEN_EVENT_STATUSR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Reports whether the effects of the last write to ITS_GEN_EVENTR are complete.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the last write to ITS_GEN_EVENTR are not guaranteed to be complete. |
| 0b1 | The effects of the last write to ITS_GEN_EVENTR are complete. |

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

Access to this field is RO.

***Accessing ITS_GEN_EVENT_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x0190 from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.10 ITS_GEN_EVENT_EIDR

The ITS_GEN_EVENT_EIDR characteristics are:

**Purpose**

ITS generate incoming event EventID register.

This register is used to specify the EventID in a request to the ITS to generate an incoming event. The DeviceID of the event is specified in the ITS_GEN_EVENT_DIDR register.

**Configuration**

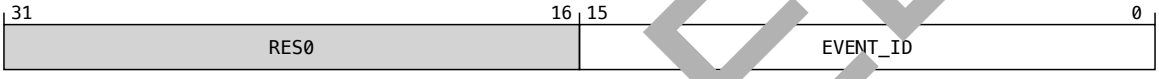
There are no configuration notes.

**Attributes**

ITS_GEN_EVENT_EIDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**EVENT_ID, bits [15:0]**

EventID in the ITS Domain containing the register.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

***Accessing ITS_GEN_EVENT_EIDR***

Accesses to this register use the following encodings:

Accessible at offset 0x0088 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_GEN_EVENT_STATUSR.IDLE != 1, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.11 ITS_GEN_EVENT_DIDR

The ITS_GEN_EVENT_DIDR characteristics are:

**Purpose**

ITS generate incoming event DeviceID register.

This register is used to specify the DeviceID in a request to the ITS to generate an incoming event. The EventID of the event is specified in the ITS_GEN_EVENT_EIDR register.

**Configuration**

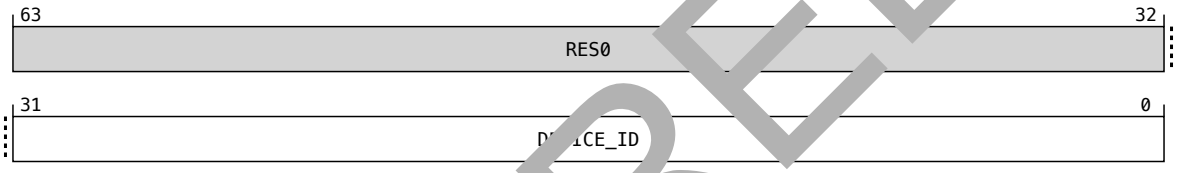
There are no configuration notes.

**Attributes**

ITS_GEN_EVENT_DIDR is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:32]**

Reserved, RES0.

**DEVICE_ID, bits [31:0]**

The DeviceID for the event generated as the result of a write to ITS_GEN_EVENTR.

The reset behavior of this field is:

- This field resets to an UNKNOWN value.

***Accessing ITS_GEN_EVENT_DIDR***

Accesses to this register use the following encodings:

Accessible at offset 0x0 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_GEN_EVENT_STATUSR.IDLE != 1, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.12 ITS_IDR0

The ITS_IDR0 characteristics are:

**Purpose**

ITS identification register 0. Contains read-only fields with information about the ITS GIC component.

**Configuration**

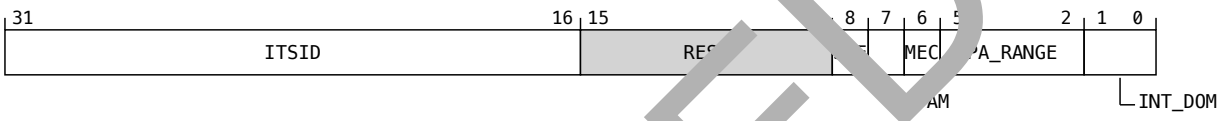
There are no configuration notes.

**Attributes**

ITS_IDR0 is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**ITSID, bits [31:16]**

Unique identifier for this ITS in the syste

This value is the same across all Interru Dom ITS.

**Bits [15:9]**

Reserved, RES0.

**SWE, bit [8]**

Software error re ting suppor the Interrupt Domain.

| SWE | Meaning |
|---|---|
| 0b0 | Software error reporting is not supported. |
| 0b1 | Software error reporting is supported. |

Sup t for Software error reporting is optional.

**MPAM, bit [**

Memory Partitioning And Monitoring (MPAM) support.

| MPAM | Meaning |
|---|---|
| 0b0 | MPAM is not supported. |
| 0b1 | MPAM is supported. |

Support for MPAM is optional.

**MEC, bit [6]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When ITS_IDR0.INT_DOM == 0b11:**

Support for Memory Encryption Contexts (MEC) for the Realm ITS Domain.

| MEC | Meaning |
|---|---|
| 0b0 | Memory Encryption Contexts are not supported. |
| 0b1 | Memory Encryption Contexts are supported. |

Support for MEC is optional.

**Otherwise:**

Reserved, RES0.

**PA_RANGE, bits [5:2]**

Physical Address range supported.

The physical address range corresponds to the syste physical ddress si

The value of this field is the same for all ITS Domains r s all ITSs the system

| PA_RA GE | Meaning |
|---|---|
| 0b0000 | 32 bits, 4GB |
| 0b0001 | 36 bits, 64GB |
| 0b0010 | 40 bits, 1TB |
| 0b0011 | 42 bits, 4TB |
| 0b0100 | 44 bits, 16TB |
| 0b0101 | 48 bits, 256TB |
| 0b0110 | 52 bits, 4PB |
| 0b0111 | 56 bits, 64PB |

V not defined above are reserved.

**INT_DOM, its [1:0]**

The ITS Domain that the register frame containing this register controls.

| INT_DOM | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing ITS_IDR0

Accesses to this register use the following encodings:

Accessible at offset `0x0000` from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.13 ITS_IDR1

The ITS_IDR1 characteristics are:

**Purpose**

ITS identification register 1. Contains read-only fields with information about the ITS GIC component.

**Configuration**

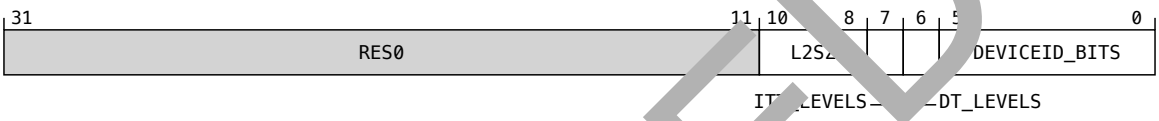
There are no configuration notes.

**Attributes**

ITS_IDR1 is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:11]**

Reserved, RES0.

**L2SZ, bits [10:8]**

Supported level 2 ITS table sizes when a 2-level table structure is used.

| L2SZ | Meaning |
|---|---|
| 0b1xx | Level 2 DT and ITT sizes supported: 64KiB |
| 0bx1x | Level 2 DT and ITT sizes supported: 16KiB |
| 0bxx1 | Level 2 DT and ITT sizes supported: 4KiB |

Values not defined above are reserved.

When both ITT_LEVELS and DT_LEVELS are 0, this field is RES0.

Arm strongly recommends that the ITS supports L2SZ value 0b111

**ITT_LEVELS, bit [7]**

Levels supported for the ITT.

| ITT_LEVELS | Meaning |
|---|---|
| 0b0 | Only a linear ITT structure is supported |
| 0b1 | Both a 2-level and a linear ITT structure is supported. |

**DT_LEVELS, bit [6]**

Levels supported for the device table.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| DT_LEVELS | Meaning |
|---|---|
| 0b0 | Only a linear device table structure is supported |
| 0b1 | Both a 2-level and a linear device table structure is supported. |

**DEVICEID_BITS, bits [5:0]**

The ITS can support translation of up to (2 ^ DEVICEID_BITS) DeviceIDs.

The maximum permitted value of this field is 32.

### *Accessing ITS_IDR1*

Accesses to this register use the following encodings:

Accessible at offset 0x0004 from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.14 ITS_IDR2

The ITS_IDR2 characteristics are:

**Purpose**

ITS identification register 2. Contains read-only fields with information about how the ITS GIC component is implemented.

**Configuration**

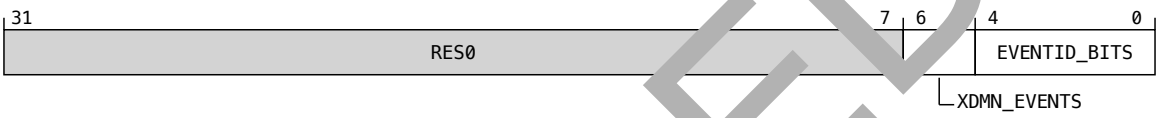
There are no configuration notes.

**Attributes**

ITS_IDR2 is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:7]**

Reserved, RES0.

**XDMN_EVENTS, bits [6:5]**

Indicates whether the ITS Domain supports translation of events associated with other Interrupt Domains.

| XDMN_EVENTS | Meaning | Applies when |
|---|---|---|
| 0b00 | The ITS Domain does not support translation of incoming events associated with other Interrupt Domains. | |
| 0b01 | The ITS Domain supports translation of incoming events associated with the Non-secure Interrupt Domain. | ITS_IDR0.INT_DOM == 0b11 |

Values not defined above are reserved.

**EVENTID_BITS, bits [4:0]**

The ITS can support translation of up to 2^(EVENTID_BITS) EventIDs.

The maximum permitted value of this field is 16.

***Accessing ITS_IDR2***

Accesses to this register use the following encodings:

Accessible at offset 0x0008 from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.15 ITS_IIDR

The ITS_IIDR characteristics are:

**Purpose**

ITS Implementer Identification Register. Provides information about the implementation and implementer of the GIC, and architecture version supported.

**Configuration**

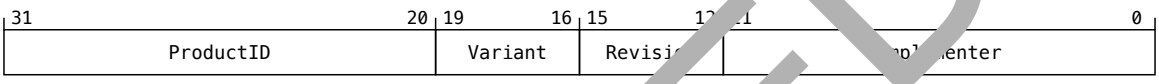
There are no configuration notes.

**Attributes**

ITS_IIDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**ProductID, bits [31:20]**

IMPLEMENTATION DEFINED value identifying the GIC part

When the ITS_PIDR0-ITS_PIDR1 registers are present, Arm expects that the ITS_PIDR0.PART_0 and ITS_PIDR1.PART_1 fields match the value of ITS_IIDR.ProductID.

If required, however, an implementation is permitted to provide values for the ITS_PIDR0.PART_0 and ITS_PIDR1.PART_1 fields that do not match the value of ITS_IIDR.ProductID

**Variant, bits [19:16]**

IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product

**Revision, bits [15:12]**

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the GIC

For an Arm implementation, the JEP106 code is 0x43B

When the CoreSight ID registers are implemented, Arm expects that the JEP106 identification code in the ITS_PIDR1-ITS_PIDR4 registers matches that reported in ITS_IIDR.

***Accessing ITS_IIDR***

Accesses to this register use the following encodings:

Accessible at offset `0x0040` from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.16 ITS_INV_DEVICER

The ITS_INV_DEVICER characteristics are:

**Purpose**

ITS cache invalidation register for a single device or a range of devices.

**Configuration**

The effects of a write to this register are not guaranteed to have completed before ITS_STATUSR.IDLE is 1.

**Attributes**

ITS_INV_DEVICER is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**I, bit [31]**

Writing 1 to this field invalidates cach[illegible] information from DTEs for the DeviceID specified in ITS_DIDR.DEVICE_ID.

| I | Meaning |
|---|---|
| 0b0 | The write has no effect. |
| 0b1 | Invalidate cached information for the specified DeviceID. |

**Bits [30:6]**

Reserved, RES0.

**EVENTID_BITS, bits [5:1]**

The range of EventIDs that the invalidation operation applies to for the specified DeviceID.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field.

The maximum value is reported in ITS_IDR2.EVENTID_BITS.

If ITS_INV_DEVICER.L1 is 1, the effective value of this field is the maximum value reported in ITS_IDR2.EVENTID_BITS.

**L1, bit [0]**

Controls which cached information the invalidation operation applies to.

When this field is 0, the operation invalidates cached information from the single level 2 DTE specified by ITS_DIDR.DEVICE_ID.

When this field is 1, the operation invalidates cached information from the level 1 DTE specified by ITS_DIDR.DEVICE_ID as well as any level 2 DTE covered by the range of DeviceIDs given by ITS_DT_CFGR.L2SZ for the DeviceID specified in ITS_DIDR.DEVICE_ID.

If ITS_DT_CFGR.STRUCTURE is 0, the effective value of this field is 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| L1 | Meaning |
|---|---|
| 0b0 | The cache invalidation operation invalidates information from the specified level 2 DTE.DEVICE_ID. |
| 0b1 | The cache invalidation operation invalidates information from the specified level 1 DTE and level 2 DTEs. |

#### *Accessing ITS_INV_DEVICER*

Accesses to this register use the following encodings:

Accessible at offset 0x0110 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_STATUS.IDLE != 1, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.17 ITS_INV_EVENTR

The ITS_INV_EVENTR characteristics are:

**Purpose**

ITS cache invalidation register for a single event or a range of events.

**Configuration**

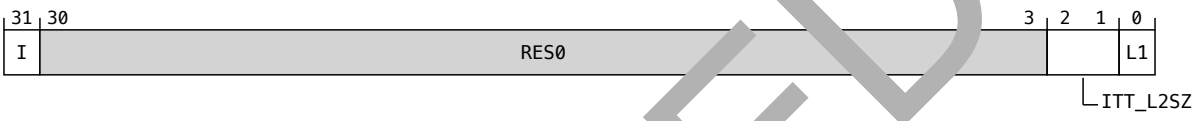
The effects of a write to this register are not guaranteed to have completed before ITS_STATUSR.IDLE is 1.

**Attributes**

ITS_INV_EVENTR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**I, bit [31]**

Writing 1 to this field invalidates cached information from ITS for the DeviceID and EventID specified in ITS_DIDR.DEVICE_ID and ITS_EIDR.EVENT_ID respectively.

| I | Meaning |
|---|---|
| 0b0 | The write has no effect. |
| 0b1 | Invalidate cached information for the specified DeviceID and EventID. |

**Bits [30:3]**

Reserved, RES0.

**ITT_L2SZ, bits [2:1]**

The range of EventIDs that the invalidation operation applies to.

| ITT_L2SZ | Meaning |
|---|---|
| 0b00 | The invalidate operation applies to cached information from any level 2 ITTE whose EventID bits [15:9] matches ITS_EIDR.EVENT_ID bits [15:9]. |
| 0b01 | The invalidate operation applies to cached information from any level 2 ITTE whose EventID bits [15:11] matches ITS_EIDR.EVENT_ID bits [15:11]. |
| 0b10 | The invalidate operation applies to cached information from any level 2 ITTE whose EventID bits [15:13] matches ITS_EIDR.EVENT_ID bits [15:13]. |

If ITS_INV_EVENTR.L1 is 0, this field is IGNORED.

Values not defined are reserved.

When programming a reserved value or an unsupported value, the ITS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**L1, bit [0]**

Controls which cached information the invalidation operation applies to.

When this field is 0, the operation invalidates cached information from the single level 2 ITTE specified by ITS_DIDR.DEVICE_ID and ITS_EIDR.EVENT_ID.

When this field is 1, the operation invalidates cached information from the level 1 ITTE specified by ITS_DIDR.DEVICE_ID and ITS_EIDR.EVENT_ID as well as any level 2 ITTE covered by the range of EventIDs given by ITS_EIDR.EVENT_ID and ITT_L2SZ for the DeviceID specified in ITS_DIDR.DEVICE_ID.

| L1 | Meaning |
|---|---|
| 0b0 | The cache invalidation operation invalidates information from the specified level 2 ITTE. |
| 0b1 | The cache invalidation operation invalidates information from the specified level 1 ITTE and level 2 ITTEs. |

### *Accessing ITS_INV_EVENTR*

Accesses to this register use the following encodings:

Accessible at offset 0x010C from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSDIS] != 0b10 && ITS_STATUSR.IDLE != 1, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.18 ITS_MEC_IDR

The ITS_MEC_IDR characteristics are:

**Purpose**

ITS MEC identification register. Contains read-only fields with information about the ITS support for MEC.

**Configuration**

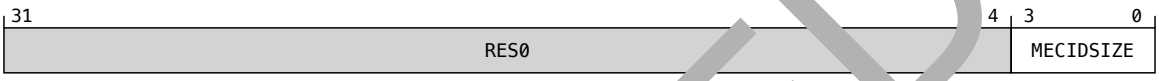
There are no configuration notes.

**Attributes**

ITS_MEC_IDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:4]**

Reserved, RES0.

**MECIDSIZE, bits [3:0]**

**When ITS_IDR0.MEC == 1:**

The number of bits minus one of MECID supported by the ITS.

The maximum permitted value is 0xF, which indicates a MECID width of 16 bits.

The value 0x0 is a valid encoding and indicates that one bit of MECID is supported.

**Otherwise:**

Reserved, RES0.

***Accessing ITS_MEC_IDR***

Accesses to this register use the following encodings:

Accessible at offset 0x01C0 from ITS_CONFIG_FRAME

- When ITS_IDR0.INT_DOM != 0b11, accesses to this register are RAZ/WI.
- When ITS_IDR0.MEC != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.19 ITS_MEC_MECID_R

The ITS_MEC_MECID_R characteristics are:

**Purpose**

ITS MEC MECID register for the Realm PAS.

**Configuration**

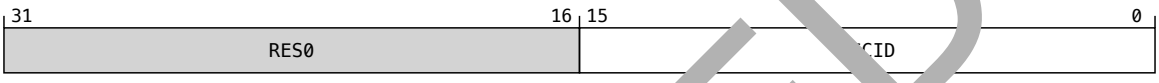
There are no configuration notes.

**Attributes**

ITS_MEC_MECID_R is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**MECID, bits [15:0]**

MECID for ITS access to Realm PA space for:

- Fetches of device table entries.
- Fetches of interrupt translation table entries.

Bits above the supported MECID size, indicated in ITS_MEC_IDR.MECIDSIZE are RES0.

If MECIDSIZE is less than 0xF, the ITS treats bits [15:MECIDSIZE+1] of this field as zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an UNKNOWN value.

***Accessing ITS_MEC_MECID_R***

Accesses to this register use the following encodings:

Accessible at offset 0x0004 from ITS_CONFIG_FRAME

- When ITS_IDR0.INT_DOM != 0b11, accesses to this register are RAZ/WI.
- When ITS_IDR0.MEC != 1, accesses to this register are RAZ/WI.
- When ITS_CR0.ITSEN == 1 or ITS_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.20 ITS_MPAM_IDR

The ITS_MPAM_IDR characteristics are:

**Purpose**

ITS MPAM identification register. Contains read-only fields with information about the ITS support for MPAM.

**Configuration**

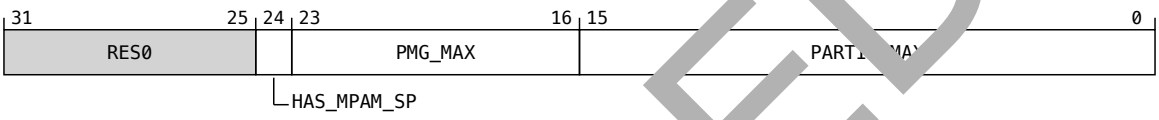
There are no configuration notes.

**Attributes**

ITS_MPAM_IDR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:25]**

Reserved, RES0.

**HAS_MPAM_SP, bit [24]**

Whether the ITS Domain has support for MPAM PARTID space selection.

If HAS_MPAM_SP is 1, the ITS uses the MPAM PARTID specified by ITS_MPAM_PARTID_R.MPAM_SP.

If HAS_MPAM_SP is 0, the following PARTID space is used for ITS accesses to memory:

- Accesses made by the Secure ITS Domain use the Secure PARTID space.
- Accesses made by the Non-secure ITS Domain use the Non-secure PARTID space.
- Accesses made by the Root ITS Domain use the Root or Secure PARTID space.
- Accesses made by the Realm ITS Domain use the Realm PARTID space.

The value of this field is the same across all ITS Domains for an ITS.

**PMG_MAX, bits [23:16]**

The maximum PMG value that is permitted to be used in the ITS Domain.

The PMG bit width is defined as the bit position of the most significant 1 in PMG_MAX[7:0], plus one, or is defined as zero if PMG_MAX is zero.

For example, if PMG_MAX == `0x0f`, the PMG bit width is 4.

This field is permitted to be zero-sized.

**PARTID_MAX, bits [15:0]**

The maximum PARTID value that is permitted to be used in the ITS Domain.

The PARTID bit width is defined as the bit position of the most significant 1 in PARTID_MAX[15:0], plus one, or is defined as zero if PARTID_MAX is zero.

For example, if PARTID_MAX == `0x0034`, the PARTID bit width is 6.

This field is permitted to be zero-sized, but Arm recommends that it is non-zero when MPAM is implemented.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing ITS_MPAM_IDR

Accesses to this register use the following encodings:

Accessible at offset `0x0200` from ITS_CONFIG_FRAME

- When ITS_IDR0.MPAM != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.21 ITS_MPAM_PARTID_R

The ITS_MPAM_PARTID_R characteristics are:

**Purpose**

ITS MPAM PARTID and PMG register.

**Configuration**

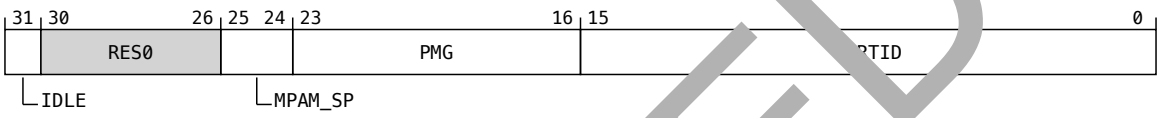
There are no configuration notes.

**Attributes**

ITS_MPAM_PARTID_R is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**IDLE, bit [31]**

Whether the effects of the previous write to this register are complete.

Following a write to this register, when this field reads-as-one, values written to this register is guaranteed to be used for subsequent memory accesses by the ITS.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of the previous write to this register are not complete. |
| 0b1 | The effects of the previous write to this register are complete. |

The reset behaviour of this field is:

On a WIC reset this field resets to '1'.

Access to this field is RO.

**Bits [30:26]**

Reserved, RES0.

**MPAM_SP, bits [25:24]**

**When ITS_MPAM_IDR.HAS_MPAM_SP == 1 and ITS_IDR0.INT_DOM == 0b00:**

MPAM PARTID space for the Secure ITS Domain.

This field is Guarded by ITS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b00 | Secure PARTID space. |
| 0b01 | Non-secure PARTID space. |

Values not defined above are reserved.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Programming a reserved value results in the ITS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When ITS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**When ITS_MPAM_IDR.HAS_MPAM_SP == 1 and ITS_IDR0.INT_DOM == 0b01:**

MPAM PARTID space for the Non-secure ITS Domain.

This field is Guarded by ITS_MPAM_PARTID_R.IDLE.

| MPAM_S | eaning |
|---|---|
| 0b0 | N -secure PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in TS usi an UNK OWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets an PLEME TION DEFINED value.

Accessing this field has the fo ing be vior:

- When ITS_MPA PARTID_R DLE = 0, access to this field is RO.
- Otherwise, ac s this field is R

**When ITS_MPAM_IDR.HAS_ PAM_SP == 1 and ITS_IDR0.INT_DOM == 0b10:**

MPAM PA TID space for the 3 ITS Domain.

This f d is Gu ed by ITS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b00 | Secure PARTID space. |
| 0b01 | Non-secure PARTID space. |
| 0b10 | Root PARTID space. |
| 0b11 | Realm PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the ITS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When ITS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When ITS_MPAM_IDR.HAS_MPAM_SP == 1 and ITS_IDR0.INT_DOM == 0b11:**

MPAM PARTID space for the Realm ITS Domain.

This field is Guarded by ITS_MPAM_PARTID_R.IDLE.

| MPAM_SP | Meaning |
|---|---|
| 0b01 | Non-secure PARTID space. |
| 0b11 | Realm PARTID space. |

Values not defined above are reserved.

Programming a reserved value results in the ITS using an UNKNOWN PARTID space.

The reset behavior of this field is:

- On a GIC reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When ITS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**Otherwise:**

Reserved, RES0.

**PMG, bits [23:16]**

PMG for accesses to memory by the ITS Domain.

Bits above the supported PMG bit width, as indicated by ITS_MPAM_IDR.PMG_MAX, are RES0.

If a value greater than ITS_MPAM_IDR.PMG_MAX is programmed, an UNKNOWN PMG is used.

This field is Guarded by ITS_MPAM_PARTID_R.IDLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0x00.

Accessing this field has the following behavior:

- When ITS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

**PARTID, bits [15:0]**

PARTID for accesses to memory by the ITS Domain.

Bits above the supported PARTID bit width, as indicated by ITS_MPAM_IDR.PARTID_MAX, are RES0.

If a value greater than ITS_MPAM_IDR.PARTID_MAX is programmed, an UNKNOWN PARTID is used.

This field is Guarded by ITS_MPAM_PARTID_R.IDLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0x0000.

Accessing this field has the following behavior:

- When ITS_MPAM_PARTID_R.IDLE == 0, access to this field is RO.
- Otherwise, access to this field is RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing ITS_MPAM_PARTID_R

Accesses to this register use the following encodings:

Accessible at offset `0x0204` from ITS_CONFIG_FRAME

- When ITS_IDR0.MPAM != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.22 ITS_READ_EVENTR

The ITS_READ_EVENTR characteristics are:

**Purpose**

ITS read event request register.

This register is used to read the translation information for an EventID and DeviceID specified in ITS_EIDR and ITS_DIDR registers, respectively.

**Configuration**

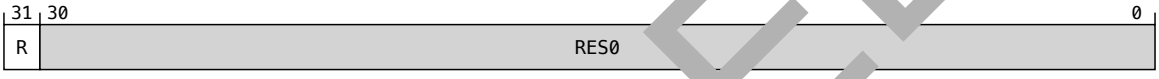
The effects of a write to this register are not guaranteed to have completed before ITS_STATUSR.IDLE is 1.

**Attributes**

ITS_READ_EVENTR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**R, bit [31]**

Request ITS to read event translation information.

| R | Meaning |
|---|---|
| 0b0 | The write has no effect on the ITS. |
| 0b1 | Read the translation information for the specified DeviceID and EventID into ITS_READ_EVENT_DATAR. |

**Bits [30:0]**

Reserved, RES0.

***Accessing ITS_READ_EVENTR***

Access to this register use the following encodings:

Accessible at offset 0x0114 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_STATUSR.IDLE != 1, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.23 ITS_READ_EVENT_DATAR

The ITS_READ_EVENT_DATAR characteristics are:

**Purpose**

ITS read event data register.

This register is used to return translation information for an EventID and DeviceID specified in ITS_EIDR and ITS_DIDR registers, respectively.

**Configuration**

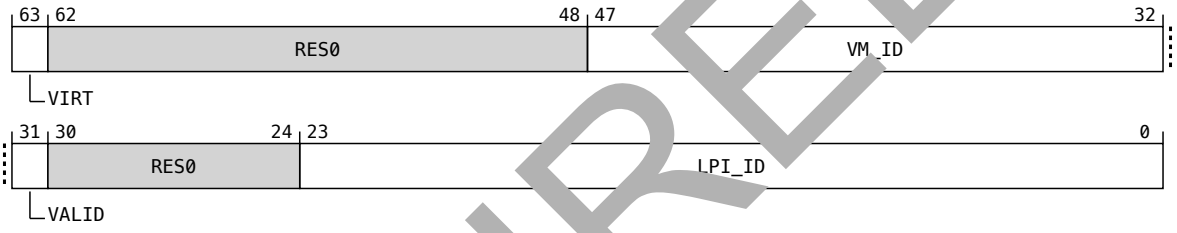
There are no configuration notes.

**Attributes**

ITS_READ_EVENT_DATAR is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**VIRT, bit [63]**

**When ITS_IDR0.INT_DOM != 0b10:**

Specifies if the interrupt message generated by the ITS to the IRS in response to this event is for a physical or a virtual interrupt.

| VIRT | Meaning |
|---|---|
| 0b0 | The interrupt message generated by the ITS is for a physical interrupt |
| 0b1 | The interrupt message generated by the ITS is for a virtual interrupt |

If VALID is 0, the value of this field is UNKNOWN.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**VM_ID, bits [47:32]**

Bits[15:0] of the VM ID passed to the IRS as part of the interrupt message targeting virtual interrupts.

If VIRT is 0, this field is RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

If VALID 0, the value of this field is UNKNOWN.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**VALID, bit [31]**

Specifies whether the ITS has valid translation information for the specified EventID and DeviceID.

| VALID | Meaning |
|---|---|
| 0b0 | The EventID does not have a valid translation. |
| 0b1 | The EventID has a valid translation. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [30:24]**

Reserved, RES0.

**LPI_ID, bits [23:0]**

Bits[23:0] of the LPI ID for the specified EventID.

If unimplemented upper bits of the LPI_ID are not zero, it is IMPLEMENTATION DEFINED whether the upper bits are treated as 0 or the interrupt message is ignored by the IRS.

If VALID 0, the value of this field is UNKNOWN.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing ITS_READ_EVENT_DATAR***

Accesses to this register use the following encodings:

Accessible at offset 0x0018 from ITS_CONFIG_FRAME

- When [ITS_CR0.IDLE, ITS_CR0.ITSEN] != 0b11 or ITS_STATUSR.IDLE != 1, accesses to this register are UNKNOWN.
- Otherwise, accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.24 ITS_STATUSR

The ITS_STATUSR characteristics are:

**Purpose**

ITS Status Register.

Reports whether the effects of the last write to all of the following registers are complete:

- ITS_INV_DEVICER.
- ITS_INV_EVENTR.
- ITS_READ_EVENTR.

**Configuration**

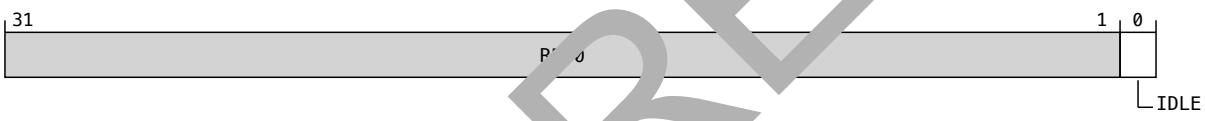
There are no configuration notes.

**Attributes**

ITS_STATUSR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME bloc

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Reports the statu of the last write all of the following registers:

- ITS_IN _DEVIC R.
- ITS_I _EVE TR.
- ITS_REA VENTR.

| IDLE | Meaning |
|---|---|
| 0 | The effects of the last write are not guaranteed to be complete. |
| 0b | The effects of the last write are guaranteed to be complete. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is RO.

***Accessing ITS_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x0120 from ITS_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.25 ITS_SWERR_STATUSR

The ITS_SWERR_STATUSR characteristics are:

**Purpose**

ITS software error status register. Specifies whether a software error has been reported. If an error is reported, it contains syndrome information for the error.

**Configuration**

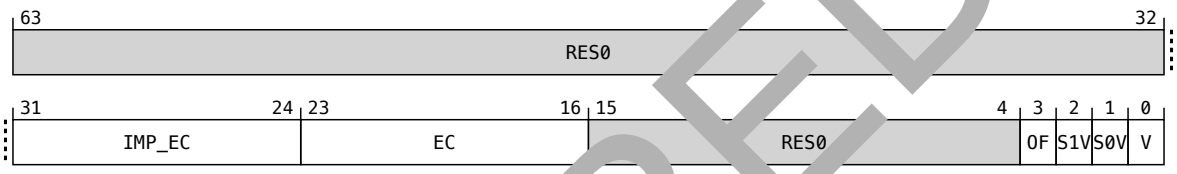
There are no configuration notes.

**Attributes**

ITS_SWERR_STATUSR is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:32]**

Reserved, RES0.

**IMP_EC, bits [31:24]**

IMPLEMENTATION DEFINED error code when ITS_SWERR_STATUS.EC == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - ITS_SWERR_STATUSR.V == 0
  - ITS_SWERR_STATUSR.EC != 0
- Otherwise access to this field is RO.

**EC, bits [23:16]**

Specifies the error code that software can use to triage and handle the error.

| EC | Meaning |
|---|---|
| 0b0 | An error was reported because of an IMPLEMENTATION DEFINED reason. |
| 0b1 | Failed lookup of L1_DTE due to an external abort. |
| 0b10 | Failed lookup of L2_DTE due to an external abort. |
| 0b11 | Failed lookup of L1_ITTE due to an external abort. |
| 0b100 | Failed lookup of L2_ITTE due to an external abort. |
| 0b101 | An incoming event could not be translated because L1_DTE.VALID is 0. |
| 0b110 | An incoming event could not be translated because L2_DTE.VALID is 0. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| EC | Meaning |
|---|---|
| 0b111 | An incoming event could not be translated because L1_ITTE.VALID is 0. |
| 0b1000 | An incoming event could not be translated because L2_ITTE.VALID is 0. |
| 0b1001 | An incoming event could not be translated because the DeviceID > (2 ^ ITS_DT_CFGR.DEVICEID_BITS) - 1. |
| 0b1010 | An incoming event could not be translated because the EventID > (2 ^ L2_DTE.EVENTID_BITS) - 1. |
| 0b1011 | An incoming event could not be translated because the DeviceID exceeds the L1_DTE.SPAN. |
| 0b1100 | An incoming event could not be translated because the EventID exceeds the L1_ITTE.SPAN. |
| 0b1101 | An incoming event could not be translated because the event is associated with the Non-secure Interrupt Domain and L2_ITTE.DAC = 0.<br>The error is reported in the Realm ITS Domain. |
| 0b1110 | ITS_GEN_EVENTR was written when ITS_GEN_EVENT_STATUSR.IDLE == 0. |
| 0b1111 | ITS_READ_EVENTR was written when ITS_STATUSR.IDLE == 0. |
| 0b10000 | ITS_INV_DEVICER was written when ITS_STATUSR.IDLE == 0. |
| 0b10001 | ITS_INV_EVENTR was written when ITS_STATUSR.IDLE == 0. |
| 0b10010 | ITS_SYNCR was written when ITS_SYNC_STATUSR.IDLE == 0. |
| 0b10011 | An incoming event could not be translated because the EventID exceeds (2 ^ ITS_IDR2.EVENTID_BITS) - |
| 0b10100 | There was a write to an ITS translation register where bits [31:16] were not all zeros. |

All other values are reserved.

The reset behavior of this field is:

- On a Global cold reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When ITS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**Bits [15:4]**

Reserved, RES0.

**OF, bit [3]**

Specifies whether multiple software errors have been detected.

When this field is 1, the syndrome information reports information about the error that last caused ITS_SWERR_STATUSR.V to transition from 0 to 1.

| OF | Meaning |
|---|---|
| 0b0 | No errors have been detected, since the error that was reported when ITS_SWERR_STATUSR.V last transitioned from 0 to 1. |
| 0b1 | At least one error has been detected, since the error that was reported when ITS_SWERR_STATUSR.V last transitioned from 0 to 1. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

When clearing ITS_SWERR_STATUSR.V to 0, if this field is nonzero, software writes 1 to clear this field to zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When ITS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

**S1V, bit [2]**

Specifies whether ITS_SWERR_SYNDROMER1 is valid.

| S1V | Meaning |
|---|---|
| 0b0 | ITS_SWERR_SYNDROMER1 is not valid. |
| 0b1 | ITS_SWERR_SYNDROMER1 is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When ITS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**S0V, bit [1]**

Specifies whether ITS_SWERR_SYNDROMER0 is valid.

| S0V | Meaning |
|---|---|
| 0b0 | ITS_SWERR_SYNDROMER0 is not valid. |
| 0b1 | ITS_SWERR_SYNDROMER0 is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When ITS_SWERR_STATUSR.V == 0, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

**V, bit [0]**

Specifies whether ITS_SWERR_STATUSR is valid and at least one software error has been reported.

| V | Meaning |
|---|---|
| 0b0 | ITS_SWERR_STATUSR is not valid. |
| 0b1 | ITS_SWERR_STATUSR is valid. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Software writes 1 to this field to clear it to zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is W1C.

#### *Accessing ITS_SWERR_STATUSR*

After reading ITS_SWERR_STATUSR, software clears the valid fields in the register to allow new errors to be reported.

However, between reading the register and clearing the valid fields, a new error might have overwritten the register.

To prevent this error being lost by software, the register prevents updates to fields that might have been updated by a new error.

This is done by ensuring a write to the register is ignored if all of the following are true:

- Any of ITS_SWERR_STATUSR.{V, OF} are nonzero before the write.
- The write does not clear the nonzero ITS_SWERR_STATUSR.{V, OF} fields to zero by writing ones to the applicable field or fields.

Accesses to this register use the following encodings:

Accessible at offset `0x0240` from ITS_CONFIG_FRAME

- When ITS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.26 ITS_SWERR_SYNDROMER0

The ITS_SWERR_SYNDROMER0 characteristics are:

**Purpose**

ITS software error syndrome register 0. Records ITS specific software error syndrome information.

**Configuration**

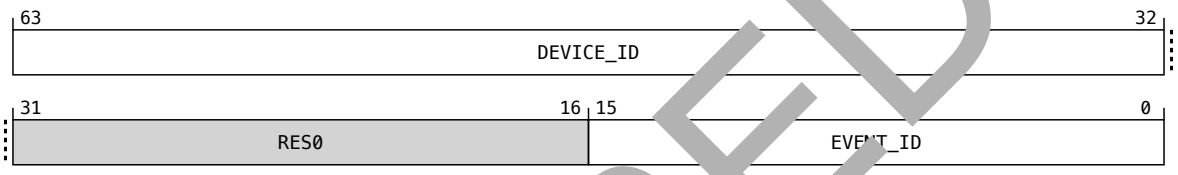
There are no configuration notes.

**Attributes**

ITS_SWERR_SYNDROMER0 is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**DEVICE_ID, bits [63:32]**

The DeviceID for the incoming event that generated the software error.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [31:16]**

Reserved, RES0.

**EVENT_ID, bits [15:0]**

The EventID for the incoming event that generated the software error.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

***Accessing ITS_SWERR_SYNDROMER0***

Accesses to this register use the following encodings:

Accessible at offset `0x0248` from ITS_CONFIG_FRAME

- When ITS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- When ITS_SWERR_STATUSR.V == 1 and ITS_SWERR_STATUSR.S0V == 1, accesses to this register are RO.
- Otherwise, accesses to this register are UNKNOWN/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.27 ITS_SWERR_SYNDROMER1

The ITS_SWERR_SYNDROMER1 characteristics are:

**Purpose**

ITS software error syndrome register 1. Records ITS specific software error syndrome information.

**Configuration**

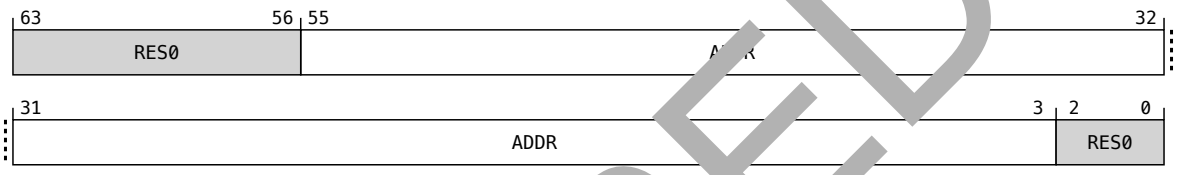
There are no configuration notes.

**Attributes**

ITS_SWERR_SYNDROMER1 is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:3]**

Bits[55:3] of the physical address of a translation structure associated with the detected error.

The address in this field is associated with the PAS of the ITS Domain where the error is detected.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in ITS_IDR0.PA_RANGE.

The reset behavior of this field is:

- On GIC reset, this field resets to an UNKNOWN value.

**Bits [2:0]**

Reserved, RES0.

***Accessing ITS_SWERR_SYNDROMER1***

Accesses to this register use the following encodings:

Accessible at offset `0x0250` from ITS_CONFIG_FRAME

- When ITS_IDR0.SWE != 1, accesses to this register are RAZ/WI.
- When ITS_SWERR_STATUSR.V == 1 and ITS_SWERR_STATUSR.S1V == 1, accesses to this register are RO.
- Otherwise, accesses to this register are UNKNOWN/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.1.28 ITS_SYNCR

The ITS_SYNCR characteristics are:

**Purpose**

ITS synchronize translation events register.

**Configuration**

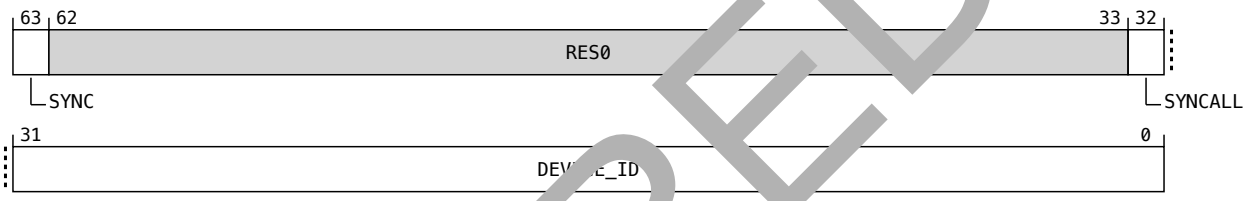
The effects of a write to this register are not guaranteed to have completed before ITS_SYNC_STATUSR.IDLE is 1.

**Attributes**

ITS_SYNCR is a 64-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**SYNC, bit [63]**

A synchronization request applies to the following Accepted events:

- Events that are generated from an IWB.
- Events that are generated from a system peripheral using an IMPLEMENTATION DEFINED mechanism.
- Events that are generated as a result of a write to ITS_GEN_EVENTR.
- Events that are generated as a result of a write to a register in an ITS translate register frame associated with the ITS Domain.

Writing 0 to this field has no effect.

| SYNC | Meaning |
|---|---|
| 0b0 | The write is IGNORED. |
| 0b1 | The write issuess a synchronization request to the ITS Domain. |

See ‘ITS synchronization requests’ for more information.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Bits [62:33]**

Reserved, RES0.

**SYNCALL, bit [32]**

Whether a synchronization request issued by writing 1 to SYNC applies to all Accepted ITS events or is only required to apply to those with specified DeviceID.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| SYNCALL | Meaning |
|---|---|
| 0b0 | Synchronize only events for the specified DeviceID. |
| 0b1 | Synchronize all events for the ITS Domain. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**DEVICE_ID, bits [31:0]**

An ITS synchronization request synchronized all Accepted ITS events with the specified DeviceID.

When SYNCALL is 1, this field is IGNORED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN valu

***Accessing ITS_SYNCR***

Accesses to this register use the following encodings:

Accessible at offset 0x0140 from ITS_CONFIG_F E

- When [ITS_CR0.IDLE, ITS_CR0.ITSE ] != 0b1 or ITS_S C_STATUSR.IDLE != 1, accesses to this register are WI.
- Otherwise, accesses to this registe are W

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.3.1.29 ITS_SYNC_STATUSR

The ITS_SYNC_STATUSR characteristics are:

**Purpose**

ITS synchronize interrupt events status register.

**Configuration**

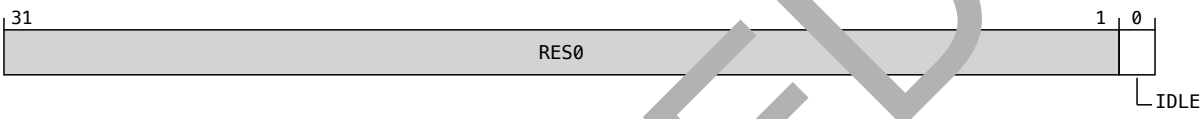
There are no configuration notes.

**Attributes**

ITS_SYNC_STATUSR is a 32-bit register.

This register is part of the ITS_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Whether the effects of the last write to ITS_SYNCR have completed.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of writing to ITS_SYNCR not guaranteed to have completed. |
| 0b1 | The effects of writing to ITS_SYNCR have completed. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Access to this field is RO.

***Accessing ITS_SYNC_STATUSR***

This register is read-only.

Accesses to this register use the following encodings:

Accessible at offset 0x0148 from ITS_CONFIG_FRAME

Accesses to this register are RO.

### 10.3.2 ITS_TRANSLATE_FRAME

The ITS_TRANSLATE_FRAME characteristics are:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## Purpose

Contains translate registers used to generate translated interrupts for an ITS Domain.

One or more ITS translate register frames are present for each supported ITS Domain.

Arm strongly recommends that translate register frame is not accessible by PEs. This is because a write to an translate register may require the ITS access to memory, leading to in-out dependencies than can potentially lead to deadlocks in the system. The translate register frames may be mapped in stage 2 translation tables shared between an SMMU and a PE, and software in a VM may perform an access to any of the mapped translate register frames. If the translate register frames are not accessible by PEs, the behavior on an attempted access from a PE is IMPLEMENTATION DEFINED and is likely to result in an External abort. If the translate register frames are accessible by PEs, writes from each PE must use a unique DeviceID that cannot spoof a write originating from a different PE or requesting device.

Each translate register frame is only accessible in the PAS associated wit[illegible] Domain.

The base address of each translate register frame is distinct from addr[illegible]es of registers [illegible]ccessible in any other PAS.

The base address of each translate register frame is aligned to 64[illegible]B.

## Attributes

ITS_TRANSLATE_FRAME is a block of size: 0x1[illegible]00 by[illegible]

## Contents

| [illegible] | Name |
|---|---|
| 0x0000 | ITS_TRANSLATER |
| 0x0008 | ITS_RL_TRANSLATER |

Direct accesses to oth[illegible]offsets in this b[illegible]k are RAZ/WI.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 10.3.2.1 ITS_TRANSLATER

The ITS_TRANSLATER characteristics are:

**Purpose**

ITS translate event register.

A write to this register generates a SET_EDGE event for the DeviceID of the agent writing to the register and the EventID specified as part of the write.

**Configuration**

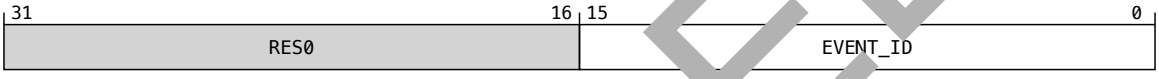
There are no configuration notes.

**Attributes**

ITS_TRANSLATER is a 32-bit register.

This register is part of the ITS_TRANSLATE_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**EVENT_ID, bits [15:0]**

The EventID to translate.

***Accessing ITS_TRANSLATER***

This register is write-only.

16-bit access to bits [15:0] of this register must be supported. When this register is written by a 16-bit transaction, bits [31:16] are written as zero.

Implementations must ensure that a unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS with a write to this register occurs in a manner that cannot be spoofed by any agent capable of performing writes.

Writes to this register are ignored if the ITS Domain is not enabled.

See Chapter 5 *Interrupt translation service (ITS)* for more information about ordering and completion requirements for writes to this register.

Accesses to this register use the following encodings:

Accessible at offset 0x0000 from ITS_TRANSLATE_FRAME

Accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.3.2.2 ITS_RL_TRANSLATER

The ITS_RL_TRANSLATER characteristics are:

**Purpose**

ITS translate event in Realm ITS Domain register.

A write to this register that uses the Non-secure PAS, generates a SET_EDGE event for the DeviceID of the agent writing to the register and the EventID specified as part of the write.

The SET_EDGE event is associated with the Non-secure Interrupt Domain and processed by the ITS in the Realm ITS Domain.

If the translation of the SET_EDGE event is successful, a SET_EDGE interrupt event is generated for the IRS in the Realm Interrupt Domain

**Configuration**

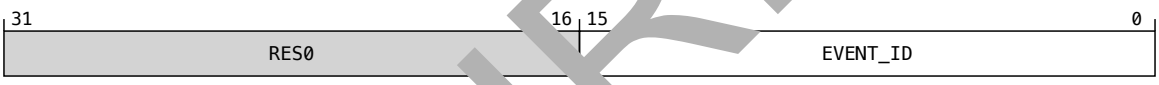
There are no configuration notes.

**Attributes**

ITS_RL_TRANSLATER is a 32-bit register.

This register is part of the ITS_TRANSLATE_FRAM[illegible].

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**EVENT_ID, bits [15:0]**

The EventID to transla[illegible].

***Accessing ITS_RL_TRANSLATER***

This regi[illegible]rite-on[illegible]

16-b[illegible] access to bits [15:0] of this register must be supported. When this register is written by a 16-bit transaction, bits [31:16] are written as zero.

Implementations must ensure that a unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS when a write to this register occurs in a manner that cannot be spoofed by any agent capable of performing writes.

Writes to this register are IGNORED if any of the following are true:

- The Non-secure ITS Domain is not enabled.
- The Realm ITS Domain is not enabled.

See Chapter 5 *Interrupt translation service (ITS)* for more information about ordering and completion requirements for writes to this register.

Accesses to this register use the following encodings:

Accessible at offset 0x0008 from ITS_TRANSLATE_FRAME

- When ITS_IDR0.INT_DOM == 0b01 and IsITSDomainImplemented(Realm), accesses to this register are WO.
- Otherwise, accesses to this register are RAZ/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# 10.4 IWB register frames

## 10.4.1 IWB_CONFIG_FRAME

The IWB_CONFIG_FRAME characteristics are:

### Purpose

Contains control registers for an IWB.

The base address is aligned to 64KB.

Access to this register frame in a PAS associated with an unimplemented Interrupt Domain is CONSTRAINED UNPREDICTABLE with a choice of:

- The access is RAZ/WI
- The access is made to an implemented Interrupt Domain as follows:
  - An access in the Realm PAS is translated to an access in the Non-secure Interrupt Domain
  - An access in the Secure PAS is translated to an access in the Non-secure Interrupt Domain
  - An access in the Root PAS is translated to an access in the GMPPAS of the IWB

### Attributes

IWB_CONFIG_FRAME is a block of size 0x10000 bytes

### Contents

| Offset | Name |
|---|---|
| 0x0000 | IWB_IDR0 |
| 0x0040 | IWB_IIDR |
| 0x0044 | IWB_AIDR |
| 0x0080 | IWB_CR0 |
| 0x00C0 | IWB_WENABLE_STATUSR |
| 0x00C4 | IWB_WDOMAIN_STATUSR |
| 0x00C8 | IWB_WRESAMPLER |
| 0XE00 - 0XEFC | IMPLEMENTATION DEFINED |
| 0x2000 + (4 * n) for n in 2047:0 | IWB_WENABLER<n> |
| 0x4000 + (4 * n) for n in 2047:0 | IWB_WTMR<n> |
| 0x8000 + (4 * n) for n in 4095:0 | IWB_WDOMAINR<n> |

Direct accesses to other offsets in this block are RAZ/WI.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**Note**

For information on the use of offsets 0xE00 - 0xEFC see 10.6 *CoreSight identification registers*.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.4.1.1 IWB_AIDR

The IWB_AIDR characteristics are:

**Purpose**

IWB Architecture Identification Register. Identifies the GIC architecture version to which the implementation conforms.

**Configuration**

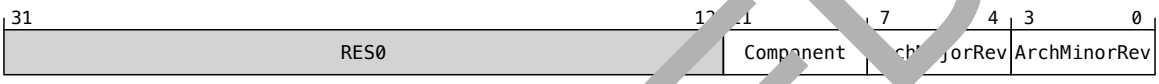
There are no configuration notes.

**Attributes**

IWB_AIDR is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:12]**

Reserved, RES0.

**Component, bits [11:8]**

GIC component

| Component | Meaning |
|---|---|
| 0b0000 | IRS |
| 0b0001 | ITS |
| 0b0010 | IWB |

**ArchMajorRev, bits [7:4]**

Major Architecture revision.

| ArchMajorRev | Meaning |
|---|---|
| 0b0000 | GICv5.x |

**ArchMinorRev, bits [3:0]**

Minor Architecture revision.

| ArchMinorRev | Meaning |
|---|---|
| 0b0000 | GICv5.0 |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IWB_AIDR

Accesses to this register use the following encodings:

Accessible at offset `0x0044` from IWB_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.2 IWB_IDR0

The IWB_IDR0 characteristics are:

**Purpose**

IWB ID register 0. Contains read-only fields with information about the IWB GIC component.

**Configuration**

There are no configuration notes.

**Attributes**

IWB_IDR0 is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:15]**

Reserved, RES0.

**INT_DOMS, bits [14:11]**

The Interrupt Domains supported by the IWB

| INT_DOMS | Meaning |
|---|---|
| 0b0001 | Only the Secure Interrupt Domain is supported |
| 0b0010 | Only the Non-secure Interrupt Domain is supported |
| 0b0111 | The EL3, Secure and Non-secure Interrupt Domains are supported |
| 0b1110 | The EL3, Realm, and Non-secure Interrupt Domains are supported |
| 0b1111 | The EL3, Realm, Secure, and Non-secure Interrupt Domains are supported |

All values not listed are reserved.

**IW_RANGE, bits [10:0]**

Indicates the number of implemented wire control registers.

The number is reported as the highest numbered wire control field as multiple of 32 minus one.

For example:

- A value of 0 means that the IWB supports register accesses to control wires 0-31
- A value of 1 means that the IWB supports register accesses to control wires 0-63
- . . . and so on until 0-65535.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IWB_IDR0

Accesses to this register use the following encodings:

Accessible at offset `0x0000` from IWB_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.3 IWB_IIDR

The IWB_IIDR characteristics are:

**Purpose**

IWB Implementer Identification Register. Provides information about the implementation and implementer of the GIC, and architecture version supported.

**Configuration**

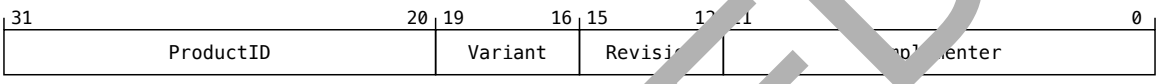
There are no configuration notes.

**Attributes**

IWB_IIDR is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**ProductID, bits [31:20]**

IMPLEMENTATION DEFINED value identifying the GIC part

When the IWB_PIDR0-IWB_PIDR1 registers are present, Arm expects that the IWB_PIDR0.PART_0 and IWB_PIDR1.PART_1 fields match the value of IWB_IIDR.ProductID.

If required, however, an implementation is permitted to provide values for the IWB_PIDR0.PART_0 and IWB_PIDR1.PART_1 fields that do not match the value of IWB_IIDR.ProductID

**Variant, bits [19:16]**

IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product

**Revision, bits [15:12]**

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product

**Implementer, bits [11:**

Contains the JEP106 code of the company that implemented the GIC

For an Arm implementation, the JEP106 code is 0x43B

When the CoreSight ID registers are implemented, Arm expects that the JEP106 identification code in the IWB_PIDR1-IWB_PIDR4 registers matches that reported in IWB_IIDR.

***Accessing IWB_IIDR***

Accesses to this register use the following encodings:

Accessible at offset `0x0040` from IWB_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.4 IWB_CR0

The IWB_CR0 characteristics are:

**Purpose**

IWB control register 0.

**Configuration**

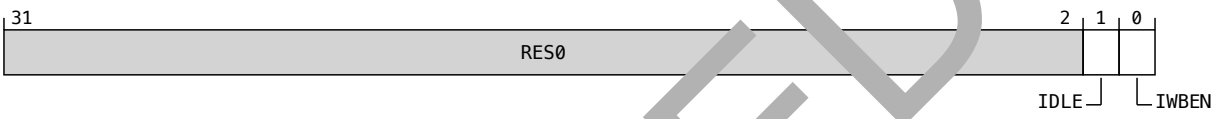
There are no configuration notes.

**Attributes**

IWB_CR0 is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:2]**

Reserved, RES0.

**IDLE, bit [1]**

Whether the transition between enabled and disabled states of the IWB is complete.

| IDLE | Meaning |
|---|---|
| 0b0 | The effects of updating IWBEN are not guaranteed to have completed. |
| 0b1 | The effects of updating IWBEN have completed. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Accesses to this field is RO.

**IWBEN, bit [0]**

Controls if the IWB is enabled.

This field is Guarded by IWB_CR0.IDLE.

| IWBEN | Meaning |
|---|---|
| 0b0 | Disabled. The IWB does not generate any events its destination. |
| 0b1 | Enabled. The IWB may generate events to its destination. |

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing this field has the following behavior:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- When IWB_CR0.IDLE == 0, access to this field is RO.
- When !IsAccessIWBMPPAS(), access to this field is RO.
- Otherwise, access to this field is RW.

#### *Accessing IWB_CR0*

Accesses to this register use the following encodings:

Accessible at offset `0x0080` from IWB_CONFIG_FRAME

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.5 IWB_WDOMAIN_STATUSR

The IWB_WDOMAIN_STATUSR characteristics are:

**Purpose**

IWB wire assignment status register for an Interrupt Domain.

**Configuration**

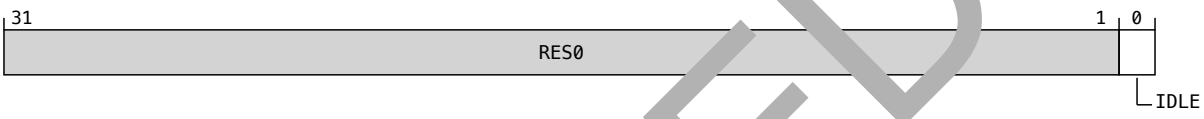
There are no configuration notes.

**Attributes**

IWB_WDOMAIN_STATUSR is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Tracks status of writes to IWB_WDOMAINR<n> on this IWB.

| IDLE | Meaning |
|---|---|
| 0b0 | A write to IWB_WDOMAINR<n> is in progress. |
| 0b1 | No write to IWB_WDOMAINR<n> is in progress. |

***Accessing IWB_WDOMAIN_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x4 from IWB_CONFIG_FRAME

- When IsAccessIWBMPPAS(), accesses to this register are RO.
- Otherwise, accesses to this register are RAZ/WI.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.6 IWB_WDOMAINR<n>, n = 0 - 4095

The IWB_WDOMAINR<n> characteristics are:

**Purpose**

IWB wire Interrupt Domain selection register. Allows software to configure the Interrupt Domain that wires n * 16 through ((n * 16) + 15) are assigned to.

**Configuration**

The number of implemented IWB_WDOMAINR<n> registers is (IWB_IDR0.IW_RANGE + 1) * 2.

The effects of a write to this register are not guaranteed to have completed until IWB_WDOMAIN_STATUSR.IDLE is 1.

**Attributes**

IWB_WDOMAINR<n> is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***

| 31 30 | 29 28 | 27 26 | 25 24 | 23 22 | 21 20 | 19 18 | 17 16 | 15 14 | 13 12 | 11 10 | 9 8 | 7 6 | 5 4 | 3 2 | 1 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| WDOM15 | WDOM14 | WDOM13 | WDOM12 | WDOM11 | WDOM10 | WDOM9 | WDOM8 | WDOM7 | WDOM6 | WDOM5 | WDOM4 | WDOM3 | WDOM2 | WDOM1 | WDOM0 |

**WDOM<x> , bits [2x+1:2x], for x = 15 to**

Configures the Interrupt Domain that wire ((16 * n) + x) is assigned to.

Programming an Interrupt Domain not supported by the IWB results in CONSTRAINED UNPREDICTABLE behavior with the following options:

- No signals are generated by the IWB for a wire assigned to an unsupported Interrupt Domain.
- The wire is treated as being assigned to another supported Interrupt Domain and a read of this field returns the Interrupt Domain that wire is assigned to.

Access to control fields for wires which are not implemented are RAZ/WI.

| WDOM<x> | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When IsWireDomainRO((n * 16) + x), access to this field is RO.
- Otherwise, access to this field is RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IWB_WDOMAINR<n>

When the Interrupt Domain assigned of a wire is fixed, access to the corresponding field is **RO**.

This register can only be accessed through the MPPAS of the IWB.

Accesses to this register use the following encodings:

Accessible at offset `0x8000 + (4 * n)` from IWB_CONFIG_FRAME

- When IWB_CR0.IDLE == 0, accesses to this register are RO.
- When !IsAccessIWBMPPAS(), accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.7 IWB_WENABLE_STATUSR

The IWB_WENABLE_STATUSR characteristics are:

**Purpose**

IWB wire enable status register for an Interrupt Domain.

**Configuration**

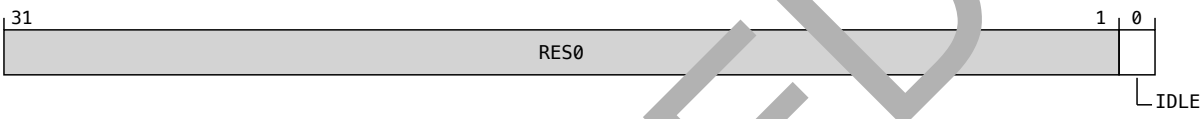
There are no configuration notes.

**Attributes**

IWB_WENABLE_STATUSR is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:1]**

Reserved, RES0.

**IDLE, bit [0]**

Tracks status of writes to IWB_WENABLER<n> on this IWB where the writes used the same PAS that was used to access this register.

| IDLE | Meaning |
|---|---|
| 0b0 | A write to IWB_WENABLER<n> using the PAS that was used to access this register is in progress. |
| 0b1 | No write to IWB_WENABLER<n> using the PAS that was used to access this register is in progress. |

***Accessing IWB_WENABLE_STATUSR***

Accesses to this register use the following encodings:

Accessible at offset 0x00C0 from IWB_CONFIG_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.8 IWB_WENABLER<n>, n = 0 - 2047

The IWB_WENABLER<n> characteristics are:

**Purpose**

IWB wire enable register. Allows software to configure if individual wires are enabled or disabled.

**Configuration**

The number of implemented IWB_WENABLER<n> registers is IWB_IDR0.IW_RANGE + 1.

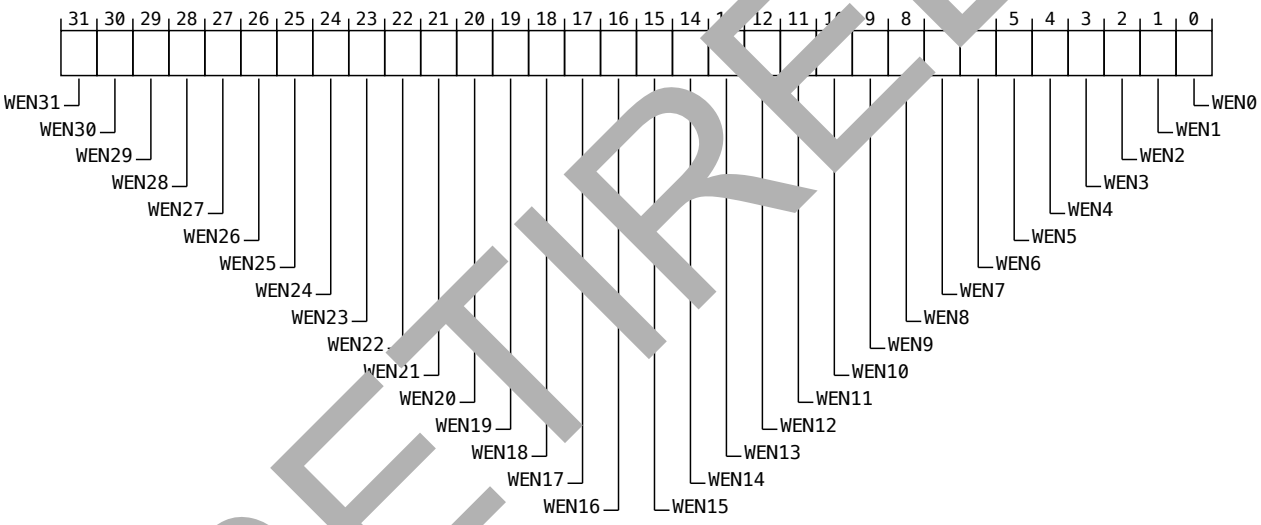
The effects of a write to this register are not guaranteed to have completed before IWB_WENABLE_STATUSR.IDLE is 1.

**Attributes**

IWB_WENABLER<n> is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**WEN<x> , bits[x], for x = 31 to 0**

Configures if wire (32 * n) + x) is enabled or disabled.

Access to control fields for wires which are not implemented are RAZ/WI.

| WEN<x> | Meaning |
|---|---|
| 0b0 | Wire disabled |
| 0b1 | Wire enabled |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- When !IsWireAccessible(n, x), access to this field is RAZ/WI.
- Otherwise, access to this field is RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing IWB_WENABLER<n>

When accessed using the MPPAS of the IWB, all fields are RW.

When accessed using any other PAS, a field is RW if all of the following are true:

- The wire corresponding to the field is assigned to an Interrupt Domain that is implemented by the IWB.
- The PAS corresponding to the access is same as the PAS associated with the Interrupt Domain to which the wire is assigned.

Otherwise, the field is RAZ/WI for that access.

Accesses to this register use the following encodings:

Accessible at offset `0x2000 + (4 * n)` from IWB_CONFIG_FRAME

- When IWB_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 10.4.1.9 IWB_WRESAMPLER

The IWB_WRESAMPLER characteristics are:

**Purpose**

IWB wire resample register.

**Configuration**

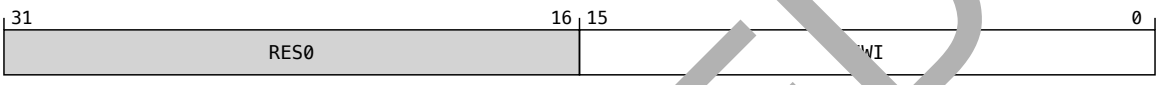
There are no configuration notes.

**Attributes**

IWB_WRESAMPLER is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**Bits [31:16]**

Reserved, RES0.

**IWI, bits [15:0]**

Input Wire Index. Specifies the wire to resample.

Following a write to this register, if all of the following are true, the wire is resampled:

- The access to this register is performed using the PAS associated with the Interrupt Domain that the wire is assigned to or the MPAS of the IWD.
- The specified wire is enabled.

If the specified wire is disabled, the write to this register has no effect.

See 'IWB wire control registers' for more information.

The reset behavior of this field is:

On a PIC reset this field resets to an UNKNOWN value.

***Accessing IWB_WRESAMPLER***

Accesses to this register use the following encodings:

Accessible at offset 0x00C8 from IWB_CONFIG_FRAME

- When IWB_CR0.IDLE == 0, accesses to this register are WI.
- Otherwise, accesses to this register are WO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.4.1.10 IWB_WTMR<n>, n = 0 - 2047

The IWB_WTMR<n> characteristics are:

**Purpose**

IWB Wire Trigger mode register. Allows software to configure if the wire signal is level-sensitive or edge-triggered.

**Configuration**

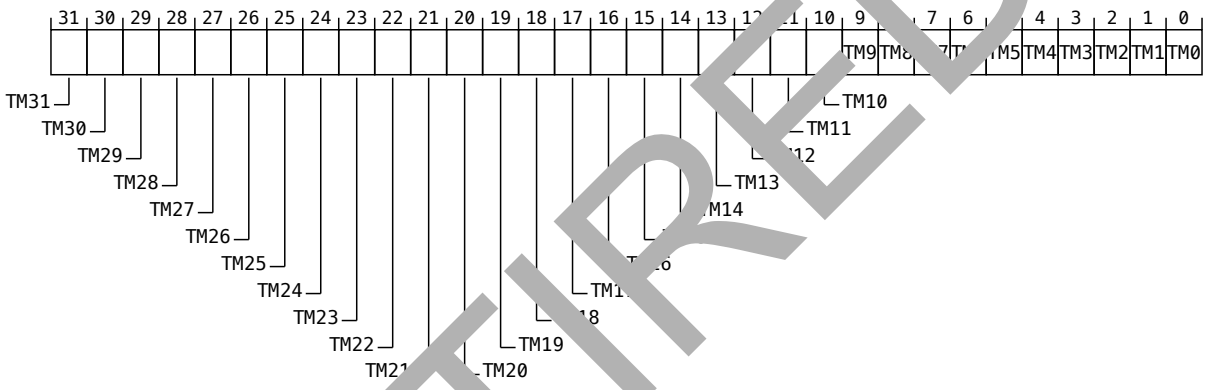
The number of implemented IWB_WTMR<n> registers is IWB_IDR0.IW_RANGE + 1.

**Attributes**

IWB_WTMR<n> is a 32-bit register.

This register is part of the IWB_CONFIG_FRAME block.

***Field descriptions***




**TM<x> , bits [x], for = 31 to 0**

Configures if wire ((32 n) + x) is level-sensitive or edge-triggered.

When the wi is configured as level-sensitive, a CLEAR event is sent to the ITS when the wire signal is de-as d, and SET_LE EL event is sent to the ITS when the wire signal is asserted.

When the ire is co ured as edge-triggered, only SET_EDGE event are generated to the ITS and only when the ire changes from de-asserted to asserted.

W sWireConfigRO(n * 32 + x) is 0, this field resets to 0.

Acces control fields for wires which are not implemented are RAZ/WI.

| TM<x> | Meaning |
|---|---|
| 0b0 | Edge-triggered |
| 0b1 | Level-sensitive |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RO if any of the following are true:
  - !IsWireAccessible(n, x)

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

    - IsWireConfigRO((n * 32) + x)
- Otherwise, access to this field is RW.

#### *Accessing IWB_WTMR<n>*

When the Trigger mode of a wire is software programmable, all of the following are true:

- When accessed using the MPPAS of the IWB, all fields are RW.
- When accessed using any other PAS, a field is RW if all of the following are true:
  - The wire corresponding to the field is assigned to an Interrupt Domain that is implemented by the IWB.
  - The PAS corresponding to the access is same as the PAS associated with the Interrupt Domain to which the wire is assigned.

Otherwise, the field is **RO** for that access.

Accesses to this register use the following encodings:

Accessible at offset `0x4000 + (4 * n)` from IWB_CONFIG_FRAME

- When IWB_CR0.IDLE == 0, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# 10.5 GIC PMU register frame

## 10.5.1 GIC_PMU_FRAME

The GIC_PMU_FRAME characteristics are:

### Purpose

Contains GIC PMU registers.

The GIC PMU register frame is accessible in each PAS corresponding to an Interrupt Domain that the PMU can count events for.

The base address of the GIC PMU register frame is distinct from any other GIC register frame.

The base address of the GIC PMU register frame is aligned to 64KB.

### Attributes

GIC_PMU_FRAME is a block of size: 0x10000 bytes

### Contents

| Offset | Name |
|---|---|
| 0x[illegible]0 + (8 *<br>↪)**for** n<br>↪6[illegible] | GIC_PMEVTYPER<n> |
| 0x[illegible]800 + [illegible]8 *<br>↪[illegible]**for** n [illegible]n<br>↪63 | GIC_PMEVFILT2R<n> |
| 0xA00 + (8 *<br>↪n)**for** n in<br>↪63:0 | GIC_PMEVFILTR<n> |
| 0xD80 | GIC_PMIDR0 |

Dire[illegible] accesses[illegible] other of[illegible]s in this block are RAZ/WI.

IKCDRK The G[illegible] PM[illegible] frame d[illegible]scribes the register frame of a GIC PMU compliant with the register formats and layouts defined [illegible]the *Arm® CoreSight™ Architecture Performance Monitoring Unit Architecture*[10].

Not all registe[illegible]are shown. Only those registers where the ITS architecture specifies an architected behavior of an IMPLEMENTATION DEFINED field as specified in [10] are shown. These registers are renamed as their behavior is specified by the GICv5 architecture. For the remaining offsets and register definitions, see [10].

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 10.5.1.1 GIC_PMEVFILT2R<n>, n = 0 - 63

The GIC_PMEVFILT2R<n> characteristics are:

**Purpose**

GIC PMU Event Filter 2 Register

**Configuration**

There are no configuration notes.

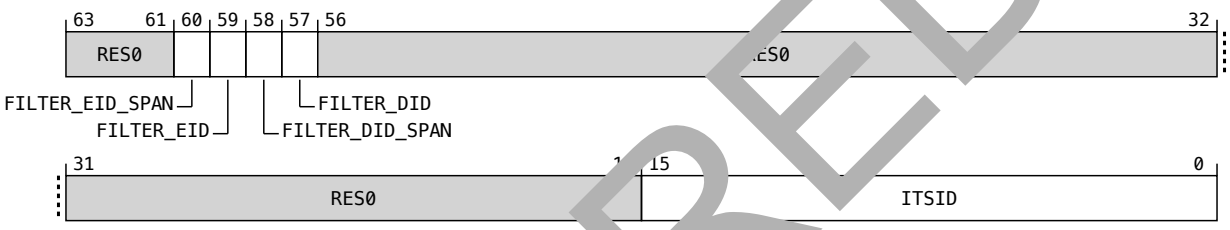
**Attributes**

GIC_PMEVFILT2R<n> is a 64-bit register.

This register is part of the GIC_PMU_FRAME block.

***Field descriptions***

**When GIC_PMEVTYPER<n>.PMEVTYPE IN {0b01x}:**




**Bits [63:61]**

Reserved, RES0.

**FILTER_EID_SPAN, bit [60]**

**When GIC_PMEVTYPER<n>.[illegible] == 1, GIC_PMEVTYPER<n>.FSPAN == 1, and FILTER_EID == 1:**

Control[illegible]f filte[illegible]g using EventID uses exact matching or matches a range of values.

[illegible] ITS [illegible]g monit[illegible]ed only implements support for 1 bit of EventID, this field is treated as 0.

See '[illegible]C Perfo[illegible]ce Monitoring Units' for more information.

| FILTER_E[illegible]SPAN | Meaning |
|---|---|
| 0b0 | GIC_PMEVFILTR<n>.EVENT_ID filters EventID on an exact match. |
| 0b1 | GIC_PMEVFILTR<n>.EVENT_ID filters EventID on a range of values. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_EID, bit [59]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When GIC_PMEVTYPER<n>.FS == 1 and FILTER_DID == 1:**

Controls whether filtering using EventID is enabled.

Filtering on EventID is only supported when also filtering on DeviceID.

See ‘GIC Performance Monitoring Units’ for more information.

| FILTER_EID | Meaning |
|---|---|
| 0b0 | Count events from all EventIDs for the matched DeviceIDs. |
| 0b1 | Count events that match the DeviceID and EventID filter programming. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_DID_SPAN, bit [58]**

**When GIC_PMEVTYPER<n>.FS == 1, GIC_PMEVTYPER<n>.FSPAN == 1, and FILTER_DID == 1:**

Controls if filtering using DeviceID uses exact matching or matches a range of values.

If the ITS being monitored only implements support for 1 bit of DeviceID, this field is treated as 0.

See ‘GIC Performance Monitoring Units’ for more information.

| FILTER_DID_SPAN | Meaning |
|---|---|
| 0b0 | GIC_PMEVFILTR<n>.DEVICE_ID filters DeviceID on an exact match. |
| 0b1 | GIC_PMEVFILTR<n>.DEVICE_ID filters DeviceID on a range of values. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_DID, bit [57]**

**When GIC_PMEVTYPER<n>.FS == 1:**

Controls whether filtering using DeviceID is enabled.

| FILTER_DID | Meaning |
|---|---|
| 0b0 | Count events from all DeviceIDs. |
| 0b1 | Count events that match the DeviceID filter programming. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [56:16]**

Reserved, RES0.

**ITSID, bits [15:0]**

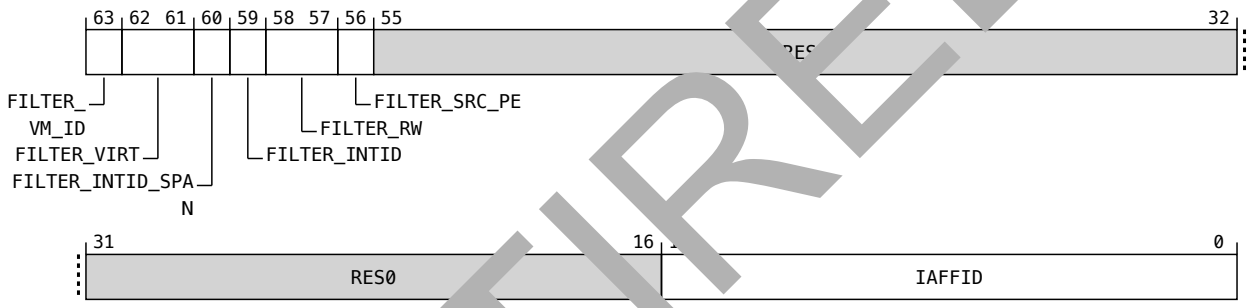
Events from the ITS that match this value are counted.

If the agent being monitored does not contain more than one ITS, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**When GIC_PMEVTYPER<n>.PMEVTYPE IN {0b00x}:**




**FILTER_VM_ID, bit [63]**

**When GIC_PMEVTYPER<n>.FS == 1 and FILTER_VIRT == 0b01:**

Controls whether counting of virtual events is filtered based on their VM ID.

When an event is selected that does not support filtering on virtual events, this field is RES0 and has no impact on the counting of events.

| FILTER_VM_ID | Meaning |
|---|---|
| 0b0 | Events translated to virtual interrupt events are not filtered on VM ID. |
| 0b1 | Events translated to virtual interrupt events are only counted if the VM ID matches the value specified in GIC_PMEVFILTR<n>.VM_ID. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_VIRT, bits [62:61]**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**When GIC_PMEVTYPER<n>.FS == 1:**

Controls whether counting of events is filtered based on whether they are virtual or physical.

When an event is selected that does not support filtering on virtual events, this field is RES0 and has no impact on the counted events.

| FILTER_VIRT | Meaning |
|---|---|
| 0b00 | Physical and virtual interrupt events are counted. |
| 0b01 | Only virtual interrupt events are counted. |
| 0b10 | Only physical interrupt events are counted. |

Values not defined are reserved.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_INTID_SPAN, bit [60]**

**When GIC_PMEVTYPER<n>.FS == 1, GIC_PMEVTYPER<n>.FSPAN == 1, and FILTER_INTID == 1:**

Controls if filtering using INTID uses exact matching or matches a range of INTID.ID values.

When an event is selected that does not support filtering on INTID, this field is RES0 and has no impact on the counted events.

See 'GIC Performance Monitoring Units' for more information.

| FILTER_INTID_SPAN | Meaning |
|---|---|
| 0b0 | GIC_PMEVFILTR<n>.ID filters INTID.ID on an exact match. |
| 0b1 | GIC_PMEVFILTR<n>.ID filters INTID.ID on a range of values. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_INTID, bit [59]**

**When GIC_PMEVTYPER<n>.FS == 1:**

Controls whether filtering using INTID is enabled.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

When an event is selected that does not support filtering on INTID, this field is RES0 and has no impact on the counted events.

| FILTER_INTID | Meaning |
|---|---|
| 0b0 | Count interrupt events for all INTIDs. |
| 0b1 | Count interrupt events that match the GIC_PMEVFILTR<n>.INTID filter programming. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_RW, bits [58:57]**

**When GIC_PMEVTYPER<n>.FS == 1:**

Controls whether counting of events is filtered based on whether they are read or write events.

When an event is selected that does not support filtering on reads or writes, this field is RES0 and has no impact on the counted events.

| FILTER_RW | Meaning |
|---|---|
| 0b00 | Both read and write events are counted. |
| 0b01 | Only read events are counted. |
| 0b10 | Only write events are counted. |

Values not defined are reserved.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**FILTER_SRC_PE, bit [56]**

**When GIC_PMEVTYPER<n>.FS == 1:**

Controls whether filtering of PE commands using source PE IAFFID is enabled.

When an event is selected that does not support filtering on source PE IAFFID, this field is RES0 and has no impact on the counted events.

| FILTER_SRC_PE | Meaning |
|---|---|
| 0b0 | Count PE commands from all PEs. |
| 0b1 | Count PE commands sent from PEs whose IAFFID match the GIC_PMEVFILTR<n>.IAFFID filter value. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [55:16]**

Reserved, RES0.

**IAFFID, bits [15:0]**

**When GIC_PMEVTYPER<n>.FS == 1 and FILTER_SRC_PE == 1:**

When filtering on source PE IAFFID, only PE commands from a PE whose IAFFID matches this value are counted.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

### *Accessing GIC_PMEVFILT2R<n>*

Accesses to this register use the following encodings:

Accessible at offset `0x800 + (8 * n)` from GIC_PMCFGR

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.5.1.2 GIC_PMEVFILTR<n>, n = 0 - 63

The GIC_PMEVFILTR<n> characteristics are:

**Purpose**

GIC PMU Event Filter Register

**Configuration**

There are no configuration notes.

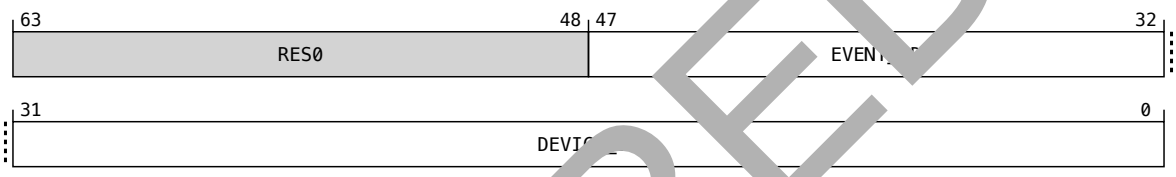
**Attributes**

GIC_PMEVFILTR<n> is a 64-bit register.

This register is part of the GIC_PMU_FRAME block.

***Field descriptions***

**When GIC_PMEVTYPER<n>.PMEVTYPE IN {0b01x}:**




**Bits [63:48]**

Reserved, RES0.

**EVENT_ID, bits [47:32]**

When filtering by EventID, the EventIDs that match this value are counted.

When GIC_PMEVFILT2R<n>.FILTER_EID is 0, this field is IGNORED.

When GIC_PMEVFILT2R<n>.FILTER_EID is 1, GIC_PMEVFILT2R<n>.FILTER_EID_SPAN controls how the EventID is matched against this value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**DEVICE_ID, bits [31:0]**

When filtering by DeviceID, the DeviceIDs that match this value are counted.

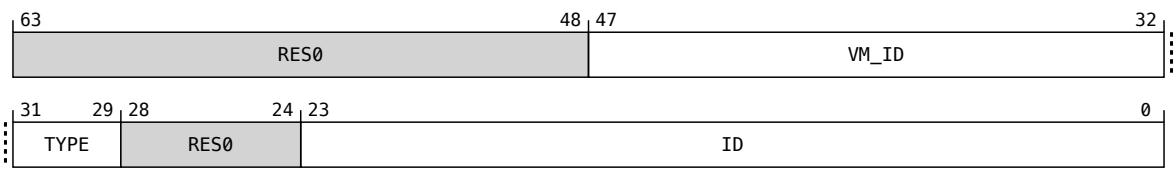
When GIC_PMEVFILT2R<n>.FILTER_DID is 0, this field is IGNORED.

When GIC_PMEVFILT2R<n>.FILTER_DID is 1, GIC_PMEVFILT2R<n>.FILTER_DID_SPAN controls how the DeviceID is matched against this value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**When GIC_PMEVTYPER<n>.PMEVTYPE IN {0b00x}:**




RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Bits [63:48]**

Reserved, RES0.

**VM_ID, bits [47:32]**

**When GIC_PMEVFILT2R<n>.FILTER_VM_ID == 1:**

When filtering on VM_ID only events that are translated to virtual interrupt events matching this VM_ID are counted.

When an event is selected that does not support filtering on VM ID, this field is RES0 and has no impact on the counted events.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**TYPE, bits [31:29]**

**When GIC_PMEVFILT2R<n>.FILTER_INTID == 1:**

Type of the interrupt.

When an event is selected that does not support filtering on INTID, this field is RES0 and has no impact on the counted events.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

The reset behavior of this field is:

On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [28:24]**

Reserved, RES0.

**ID, bits [23:0]**

**When GIC_PMEVFILT2R<n>.FILTER_INTID == 1:**

The ID of the interrupt.

ID and TYPE together form an INTID.

The monitored IRS may implement fewer than 24 bits of ID. Unimplemented upper bits are RES0.

When GIC_PMEVFILT2R<n>.FILTER_INTID is 1, GIC_PMEVFILT2R<n>.FILTER_INTID_SPAN controls how the INTID is matched against this value.

When an event is selected that does not support filtering on INTID, this field is RES0 and has no impact on the counted events.

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

#### *Accessing GIC_PMEVFILTR<n>*

Accesses to this register use the following encodings:

Accessible at offset `0xA00 + (8 * n)` from GIC_PMU_FRAME

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.5.1.3 GIC_PMEVTYPER<n>, n = 0 - 63

The GIC_PMEVTYPER<n> characteristics are:

**Purpose**

GIC PMU Event Type Select Register

**Configuration**

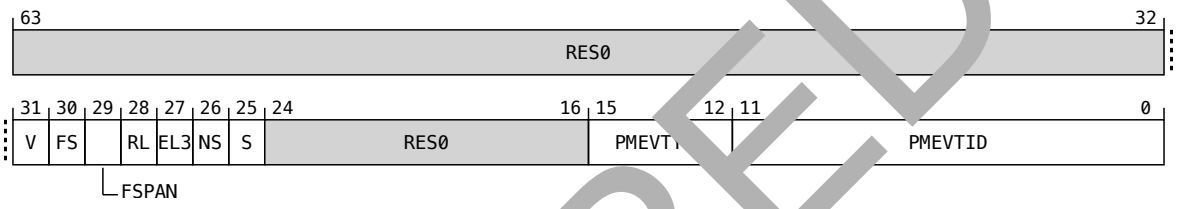
There are no configuration notes.

**Attributes**

GIC_PMEVTYPER<n> is a 64-bit register.

This register is part of the GIC_PMU_FRAME block.

***Field descriptions***




**Bits [63:32]**

Reserved, RES0.

**V, bit [31]**

Whether PMEVTYPE and PMEVTID select a valid event.

| V | Meaning |
|---|---|
| 0b0 | The event selected by PMEVTYPE and PMEVTID is invalid. |
| 0b1 | The event selected by PMEVTYPE and PMEVTID is valid. |

The reset behavior of this field is:

On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is RO.

**FS, bit [30]**

Whether filtering is supported for the event selected in GIC_PMEVTID.

When V is 0, the value of this field is UNKNOWN.

| FS | Meaning |
|---|---|
| 0b0 | Filtering for the selected event is not supported. |
| 0b1 | Filtering for the selected event is supported. |

The reset behavior of this field is:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is RO.

**FSPAN, bit [29]**

Whether filtering using a range of values is supported for the event selected in GIC_PMEVTID. See 'GIC Performance Monitoring Units' for more information about filtering on a range of values.

When V is 0 or FS is 0, the value of this field is UNKNOWN.

| FSPAN | Meaning |
|---|---|
| 0b0 | Filtering using a range of values is not supported. |
| 0b1 | Filtering using a range of values is supported. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Access to this field is RO.

**RL, bit [28]**

**When GIC_PMIDR0.OACE == 1 and GIC_PMIDR0.DOM_RL == 1:**

Configures matching the Realm Interrupt Domain.

| RL | Meaning |
|---|---|
| 0b0 | Events or characteristics attributable to the Realm Interrupt Domain are unaffected by this bit. |
| 0b1 | Do not count events or monitor characteristics attributable to the Realm Interrupt Domain. |

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Realm operation of the agent being monitored.

Accessing this bit has the following behavior:

- Reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Realm operation of the agent being monitored.
  Otherwise, this bit is read/write.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**EL3, bit [27]**

**When GIC_PMIDR0.OACE == 1 and GIC_PMIDR0.DOM_EL3 == 1:**

Configures matching the EL3 Interrupt Domain.

| EL3 | Meaning |
|---|---|
| 0b0 | Events or characteristics attributable to the EL3 Interrupt Domain are unaffected by this bit. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| EL3 | Meaning |
|---|---|
| 0b1 | Do not count events or monitor characteristics attributable to the EL3 Interrupt Domain. |

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to EL3 operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to EL3 operation of the agent being monitored.
- Otherwise, this bit is read/write.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN valu

**Otherwise:**

Reserved, RES0.

**NS, bit [26]**

**When GIC_PMIDR0.OACE == 1 and GIC_PMIDR0.DOM_NS == 1:**

Configures matching the Non-secure Interrupt Domain.

| NS | Meaning |
|---|---|
| 0b0 | Events or characteristics attributable to the Non-secure Interrupt Domain are unaffected by this bit. |
| 0b1 | Do not count events or monitor characteristics attributable to the Non-secure Interrupt Domain. |

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Non-secure operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Non-secure operation of the agent being monitored.
- Otherwise, this bit is read/write.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**S, bit [25]**

**When GIC_PMIDR0.OACE == 1 and GIC_PMIDR0.DOM_S == 1:**

Configures matching the Secure Interrupt Domain.

| S | Meaning |
|---|---|
| 0b0 | Events or characteristics attributable to the Secure Interrupt Domain are unaffected by this bit. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| S | Meaning |
|---|---|
| 0b1 | Do not count events or monitor characteristics attributable to the Secure Interrupt Domain. |

This bit is ignored by the PMU when the PMU is not allowed to observe events or characteristics attributable to Secure operation of the agent being monitored.

Accessing this bit has the following behavior:

- This bit reads-as-zero if the PMU is never allowed to observe events or characteristics attributable to Secure operation of the agent being monitored.
- Otherwise, this bit is read/write.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN valu

**Otherwise:**

Reserved, RES0.

**Bits [24:16]**

Reserved, RES0.

**PMEVTYPE, bits [15:12]**

The performance monitor event type.

The value in this field selects the ev t for th pe selected in PMEVTYPE.

Values not defined are rese ed.

| PMEVTYPE | Meaning |
|---|---|
| 0b0000 | Architected IRS event. |
| 0b0001 | IMPLEMENTATION DEFINED IRS event. |
| 0b0010 | Architected ITS event. |
| 0b0011 | IMPLEMENTATION DEFINED ITS event. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

**PMEVTID, bits [11:0]**

The performance monitor event ID.

The value in this field selects the event for the type selected in PMEVTYPE.

If the value in this field combined with the value in PMEVTYPE selects an invalid event, all other fields in this register are IGNORED.

If less than 16 bits of event ID is implemented, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### Accessing GIC_PMEVTYPER<n>

Accesses to this register use the following encodings:

Accessible at offset `0x400 + (8 * n)` from GIC_PMU_FRAME

Accesses to this register are RW.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 10.5.1.4 GIC_PMIDR0

The GIC_PMIDR0 characteristics are:

**Purpose**

GIC PMU identification register 0. Contains read-only fields with information about the GIC PMU.

**Configuration**

There are no configuration notes.

**Attributes**

GIC_PMIDR0 is a 32-bit register.

This register is part of the GIC_PMU_FRAME block.

***Field descriptions***




**Bits [31:7]**

Reserved, RES0.

**DOM_RL, bit [6]**

Whether a component in the agent being monitored supports the Realm Interrupt Domain.

| DOM_RL | Meaning |
|---|---|
| 0b0 | The Realm Interrupt Domain is not supported by any component in the agent being monitored. |
| 0b1 | The Realm Interrupt Domain is supported by a component in the agent being monitored. |

**DOM_EL3, bit [5]**

Whether a component in the agent being monitored supports the EL3 Interrupt Domain.

| DOM_EL3 | Meaning |
|---|---|
| 0b0 | The EL3 Interrupt Domain is not supported by any component in the agent being monitored. |
| 0b1 | The EL3 Interrupt Domain is supported by a component in the agent being monitored. |

**DOM_NS, bit [4]**

Whether a component in the agent being monitored supports the Non-secure Interrupt Domain.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| DOM_NS | Meaning |
|---|---|
| 0b0 | The Non-secure Interrupt Domain is not supported by any component in the agent being monitored. |
| 0b1 | The Non-secure Interrupt Domain is supported by a component in the agent being monitored. |

**DOM_S, bit [3]**

Whether a component in the agent being monitored supports the Secure Interrupt Domain.

| DOM_S | Meaning |
|---|---|
| 0b0 | The Secure Interrupt Domain is not supported by any component in the agent being monitored. |
| 0b1 | The Secure Interrupt Domain is supported by a component in the agent being monitored. |

**OACE, bit [2]**

Whether the PMU implements the observability and access control extension.

| OACE | Meaning |
|---|---|
| 0b0 | The PMU does not implement the observability and access control extension. |
| 0b1 | The PMU implements the observability and access control extension. |

**ITS_PMU, bit [1]**

Whether this PMU counts events from an ITS.

This field can be interpreted in combination with IRS_PMU to determine the GIC PMU configuration.

See 'GIC Performance Monitoring Unit (PMU)' for more information.

| ITS_PMU | Meaning |
|---|---|
| 0b0 | This PMU does not count events from an ITS. |
| 0b1 | This PMU counts events from at least one ITS. |

**IRS_PMU, bit [0]**

Whether this PMU counts events from an IRS.

This field can be interpreted in combination with ITS_PMU to determine the GIC PMU configuration.

See 'GIC Performance Monitoring Unit (PMU)' for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| IRS_PMU | Meaning |
|---|---|
| 0b0 | This PMU does not count events from an IRS. |
| 0b1 | This PMU counts events from an IRS. |

#### Accessing GIC_PMIDR0

Accesses to this register use the following encodings:

Accessible at offset 0xD80 from GIC_PMU_FRAME

Accesses to this register are RO.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 10.6 CoreSight identification registers

$R_{JTSPV}$ In the following register frames, offsets `0xFFD0-0xFFFC` are defined as read-only identification register space:

- IRS_CONFIG_FRAME
- ITS_CONFIG_FRAME
- IWB_CONFIG_FRAME

$I_{SMPQP}$ For Arm implementations, the following assignment of fields, consistent with CoreSight ID registers[11], is used:

**Table 10.178: CoreSight ID register[11] field assignments**

| Offset | Name | Field | Value | Meaning |
|---|---|---|---|---|
| 0xFFF0 | x_CIDR0, Component ID0 | [7:0] | `0x0D` | Preamble |
| 0xFFF4 | x_CIDR1, Component ID1 | [7:4] | `0xF` | CLASS |
| | | [3:0] | `0x0` | Preamble |
| 0xFFF8 | x_CIDR2, Component ID2 | [7:0] | `0x05` | Preamble |
| 0xFFFC | x_CIDR3, Component ID3 | [7:0] | `0xB1` | Preamble |
| 0xFFE0 | x_PIDR0, Peripheral ID0 | [7:0] | IMPLEMENTATION DEFINED | Bits [7:0] of the Part number |
| 0xFFE4 | x_PIDR1, Peripheral ID1 | [7:4] | IMPLEMENTATION DEFINED | Bits [3:0] of the JEP106 Designer code |
| | | [3:0] | IMPLEMENTATION DEFINED | Bits [11:8] of the Part number |
| 0xFFE8 | x_PIDR2, Peripheral ID2 | [7:4] | IMPLEMENTATION DEFINED | REVISION |
| | | [3] | `1` | JEDEC-assigned value for DES always used |
| | | [2:0] | IMPLEMENTATION DEFINED | Bits [6:4] bits of the JEP106 Designer code |
| 0xFFEC | x_PIDR3, Peripheral ID3 | [7:4] | IMPLEMENTATION DEFINED | REVAND |
| | | [3:0] | IMPLEMENTATION DEFINED | CMOD |
| 0xFFD0 | x_PIDR4, Peripheral ID4 | [7:4] | `0` | SIZE |
| | | [3:0] | IMPLEMENTATION DEFINED | JEP106 Designer continuation code |
| 0xFFD4 | x_PIDR5, Peripheral ID5 | - | RES0 | Reserved |
| 0xFFD8 | x_PIDR6, Peripheral ID6 | - | RES0 | Reserved |
| 0xFFEC | x_PIDR7, Peripheral ID7 | - | RES0 | Reserved |
| 0xFFBC | x_DEVARCH | [31:21] | `0x23b` | JEP106 continuation and identification codes |
| | | [20] | `1` | DEVARCH is present. |
| | | [19:16] | `0` | GICv5.0 |
| | | [15:0] | `0x5A19` | ARCHID, GICv5. |

Where “x” is replaced with:

- IRS for the IRS_CONFIG_FRAME
- ITS for the ITS_CONFIG_FRAME
- IWB for the IWB_CONFIG_FRAME

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

**Table 10.179: Implemented CoreSight ID registers[11] for each GICv5 system component**

| Name | IRS | ITS | IWB |
|---|---|---|---|
| x_CIDR0 | IRS_CIDR0 | ITS_CIDR0 | IWB_CIDR0 |
| x_CIDR1 | IRS_CIDR0 | ITS_CIDR1 | IWB_CIDR1 |
| x_CIDR2 | IRS_CIDR0 | ITS_CIDR2 | IWB_CIDR2 |
| x_CIDR3 | IRS_CIDR0 | ITS_CIDR3 | IWB_CIDR3 |
| x_DEVARCH | IRS_DEVARCH | ITS_DEVARCH | IWB_DEVARCH |
| x_PIDR0 | IRS_PIDR0 | ITS_PIDR0 | IWB_PIDR0 |
| x_PIDR1 | IRS_PIDR1 | ITS_PIDR1 | IWB_PIDR1 |
| x_PIDR2 | IRS_PIDR2 | ITS_PIDR2 | IWB_PIDR2 |
| x_PIDR3 | IRS_PIDR3 | ITS_PIDR3 | IWB_PIDR3 |
| x_PIDR4 | IRS_PIDR4 | ITS_PIDR4 | IWB_PIDR4 |
| x_PIDR5 | IRS_PIDR5 | ITS_PIDR5 | IWB_PIDR5 |
| x_PIDR6 | IRS_PIDR6 | ITS_PIDR6 | IWB_PIDR6 |
| x_PIDR7 | IRS_PIDR7 | ITS_PIDR7 | IWB_PIDR7 |

Offsets and fields in the CoreSight[11] ID register region outside of those defined in Table 10.178 are RES0.

Arm recommends that implementers use this scheme to provide a consistent software discovery model. If the CoreSight ID registers are not implemented, Arm recommends that all the locations in Table 10.178 are RES0.

The Designer code field for Arm-designed implementations use continuation code 0x4 and Designer code 0x3B. Non-Arm implementations that follow this CoreSight ID register layout set the Designer fields appropriate to the implementer.

$I_{LFDTN}$ For a full description of the CoreSight ID registers see the Arm CoreSight Architecture Specification [11].

$S_{SDKTH}$ A single ARCHID value is defined for GICv5 and used by all GICv5 register frames that include the CoreSight ID registers. Software can determine which type of frame is being described by reading the AIDR register, which is at the same offset in all GICv5 register frames.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter 11
# **Data structures**

RETIRED

This chapter describes the CMCv5 data structures.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 11.1 ITS Data Structures

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

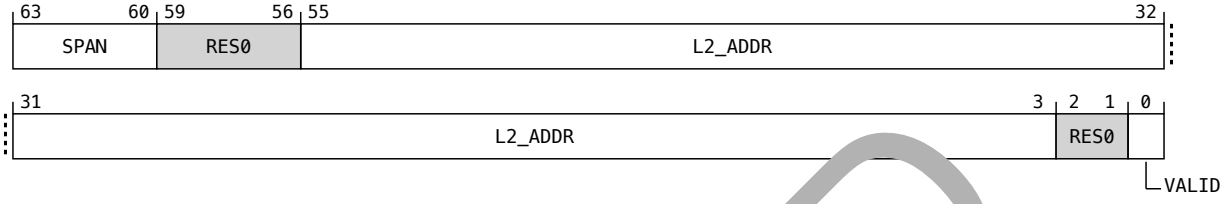
### 11.1.1 L1_DTE, Level 1 device table entry

The L1_DTE characteristics are:

#### Attributes

L1_DTE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All events identified by a DeviceID and EventID covered by this entry are ignored. |
| 0b1 | This entry is valid and L2_ADDR points to a level 2 device table. |

#### Bits [2:1]

Reserved, RES0.

#### L2_ADDR, bits [55:3]

Bits[55:3] of the address of the start of the level 2 array of device table entries.

Bits[N:0] of the resulting address are 0 where N = 2 + SPAN.

This means that the level 2 DTE array is aligned to the size of the array.

The level 2 array is accessed using the same PAS as the level 1 DTE.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in ITS_IDR0.PA_RANGE.

#### Bits [59:56]

Reserved, RES0.

#### SPAN, bits [63:60]

Size of the structure pointed to by L2_ADDR

The level 2 device table pointed to by L2_ADDR contains 2 ^ SPAN entries.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field. The maximum allowed value of this field is determined by ITS_DT_CFGR.L2SZ.

If ITS_DT_CFGR.L2SZ is 0b00, then the maximum value of this field is 9. If ITS_DT_CFGR.L2SZ is 0b01, then the maximum value of this field is 11. If ITS_DT_CFGR.L2SZ is 0b10, then the maximum value of this field is 13.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

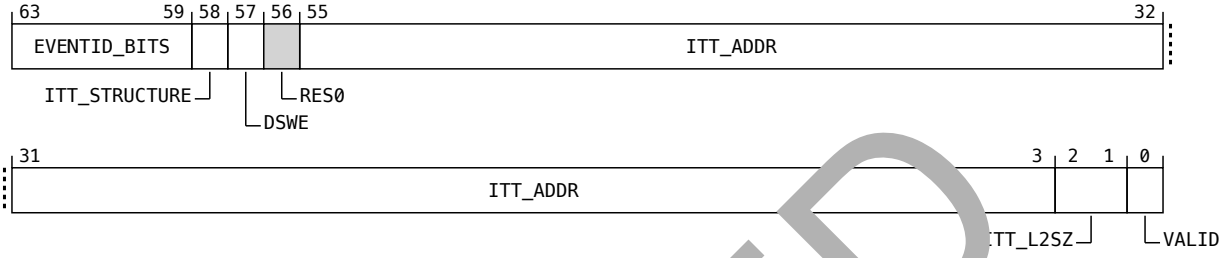
### 11.1.2 L2_DTE, Level 2 device table entry

The L2_DTE characteristics are:

#### Attributes

L2_DTE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All events identified by a DeviceID and EventID covered by this entry are ignored. |
| 0b1 | This entry is valid and ITT_ADDR points to a valid ITT. |

#### ITT_L2SZ, bits [2:1]

Level 2 ITT size when a 2-level ITT structure is used.

| ITT_L2SZ | Meaning |
|---|---|
| 0b00 | A level 2 ITT is maximum 4KB and resolves 9 bits of EventID. |
| 0b01 | A level 2 ITT is maximum 16KB and resolves 11 bits of EventID. |
| 0b10 | A level 2 ITT is maximum 64KB and resolves 13 bits of EventID. |

Values not defined above are reserved.

If EVENTID_BITS <= (9 + (2 * ITT_L2SZ)) and ITT_STRUCTURE is 1, the ITT consists of a single L1_ITTE and a single L2_ITT.

The L2_ITT contains (2 ^ EVENTID_BITS) entries.

If ITT_STRUCTURE is 1, and EVENTID_BITS < (9 + (2 * ITT_L2SZ)), it is possible to program L1_ITTE.SPAN to a value larger than EVENTID_BITS. In this case, the number of EventIDs for the DeviceID is defined by 2 ^ EVENTID_BITS, but the ITS is permitted to access all the L2_ITTEs in the range described by L1_ITTE.SPAN.

Arm recommends that ITT_STRUCTURE is 0 when EVENTID_BITS <= (9 + (2 * ITT_L2SZ)).

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

When programming a reserved value or an unsupported value, the ITS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

### ITT_ADDR, bits [55:3]

Bits[55:3] of the base physical address of the ITT for the device described by this entry.

When ITT_STRUCTURE is 0, ITT_ADDR points to a linear ITT and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N = 2 + EVENTID_BITS.
- This means that the level 2 ITTE array is aligned to the size of the array.

When ITT_STRUCTURE is 1, ITT_ADDR points to a 2-level ITT and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N is:

  Max(2, (EVENTID_BITS - (9 + (2 * ITT_L2SZ)) + 2)

- This means that the level 1 ITT is aligned to the size of the tab

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in ITS_IDR0.PA_RANGE.

### Bit [56]

Reserved, RES0.

### DSWE, bit [57]

For any ITS event generated by the device described by this entry, disable reporting of software errors for the following error codes in ITS_SWERR_STATUS.EC:

- 0x03.
- 0x04.
- 0x06.
- 0x07.
- 0x08.
- 0x0A.
- 0x0C.
- 0x0D.
- 0x
- 0x14.

| DSWE | Meaning |
|---|---|
| 0b0 | Error reporting for the specified error codes is disabled. |
| 0b1 | Error reporting for the specified error codes is enabled. |

This field is RES0, if ITS_IDR0.SWE is 0.

### ITT_STRUCTURE, bit [58]

Whether the the ITT pointed to by ITT_ADDR uses a linear or 2-level structure.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| ITT_STRUCTURE | Meaning |
|---|---|
| 0b0 | A linear ITT structure is used. |
| 0b1 | A 2-level ITT structure is used. |

If ITS_IDR1.ITT_LEVELS is 0, this field is RES0.

### EVENTID_BITS, bits [63:59]

The number of EventID bits which can be translated for this device.

The ITT pointed to by ITT_ADDR contains 2 ^ EVENTID_BITS level 2 ITT entries in total.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field. The maximum value is reported in ITS_IDR2.EVENTID_BITS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

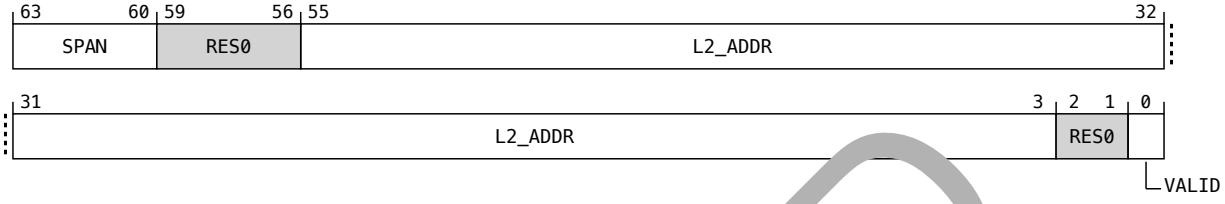
### 11.1.3 L1_ITTE, Level 1 interrupt translation table entry

The L1_ITTE characteristics are:

#### Attributes

L1_ITTE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All events identified by a DeviceID and EventID covered by this entry are ignored. |
| 0b1 | This entry is valid and L2_ADDR points to a level 2 ITT. |

#### Bits [2:1]

Reserved, RES0.

#### L2_ADDR, bits [55:3]

Bits[55:3] of the level 2 entry of ITT entries.

Bits[N:0] of the resulting address are 0 where N = 2 + SPAN.

This means that the level 2 ITTE array is aligned to the size of the array.

The level 2 array is accessed using the same PAS as the level 1 ITTE.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in ITS_IDR0.PA_RANGE.

#### Bits [59:56]

Reserved, RES0.

#### SPAN, bits [63:60]

Size of structure pointed to by L2_ADDR

The level 2 ITT pointed to by L2_ADDR contains 2 ^ SPAN entries.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field. The maximum allowed value of this field is constrained by L2DTE.ITT_L2SZ.

If L2_DTE.ITT_L2SZ is 0b00, then the maximum value of this field is 9. If L2_DTE.ITT_L2SZ is 0b01, then the maximum value of this field is 11. If L2_DTE.ITT_L2SZ is 0b10, then the maximum value of this field is 13.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

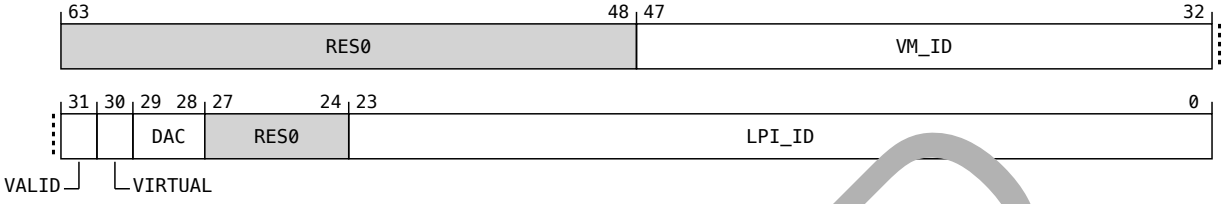
### 11.1.4 L2_ITTE, Level 2 interrupt translation table entry

The L2_ITTE characteristics are:

#### Attributes

L2_ITTE is a 8-byte structure.

#### Field descriptions




#### LPI_ID, bits [23:0]

Bits[23:0] of the LPI ID for the event described by this entry.

If unimplemented upper bits of the LPI_ID are not zero, it is IMPLEMENTATION DEFINED whether the upper bits are treated as 0 or the interrupt message is ignored by the IRS.

#### Bits [27:24]

Reserved, RES0.

#### DAC, bits [29:28]

Controls whether the event described by this entry can be associated with an Interrupt Domain different from the one associated with this ITS Domain.

| DAC | Meaning | Applies when |
|---|---|---|
| 0b00 | The event described by this entry is not permitted to be translated when it is generated by a write to a register in the PAS associated with another Domain. | |
| 0b01 | The event described by this entry is permitted to be translated when it is generated by a write to a register in the Non-secure PAS. | ITS_IDR2.XDMN_EVENTS == 0b01 |

Values not defined above are reserved.

The event is ignored and not translated if any of the following are true:

- The event is not associated with the Interrupt Domain specified by this field.
- A reserved value is programmed into this field.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### VIRTUAL, bit [30]

Controls if the interrupt message generated to the IRS is for a physical or a virtual interrupt.

| VIRTUAL | Meaning |
|---|---|
| 0b0 | The interrupt message generated by the ITS is a physical interrupt. |
| 0b1 | The interrupt message generated by the ITS is a virtual interrupt. |

When the ITT is used in the EL3 ITS Domain, this field is RES0.

### VALID, bit [31]

Interrupt translation entry is valid.

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All events identified by a DeviceID and EventID covered by this entry are ignored. |
| 0b1 | This entry is valid and events identified by a DeviceID and EventID covered by this entry are translated into events to the IRS using the VM ID and LPI ID above. |

### VM_ID, bits [47:32]

Bits[15:0] of the VM ID passed to the IRS as part of the interrupt message targeting virtual interrupts.

When Virtual is 0, this field is RES0.

### Bits [63:48]

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 11.2 IRS Data Structures

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

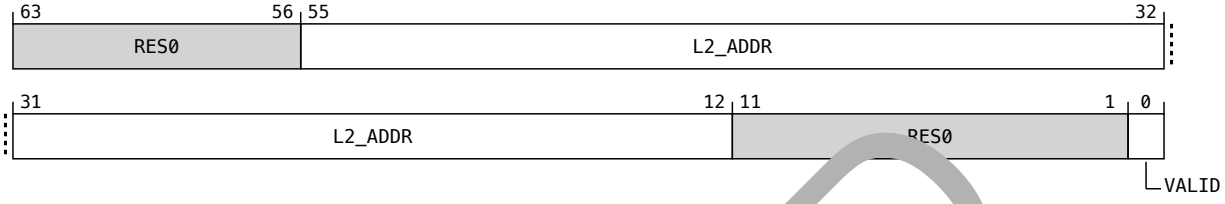
### 11.2.1 L1_VMTE, Level 1 VM table entry

The L1_VMTE characteristics are:

#### Attributes

L1_VMTE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All lookups of a VM ID covered by this entry are ignored. |
| 0b1 | This entry is valid and L2_ADDR points to a level 2 VMT. |

#### Bits [11:1]

Reserved, RES0.

#### L2_ADDR, bits [55:12]

Bits[55:12] of the address of the start of the array of level 2 VMT entries.

This means the level 2 VMT array is aligned to the size of a level 2 VMT (4KB).

The level 2 array is accessed using the same PAS as the level 1 VMTE.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

#### Bits [63:56]

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

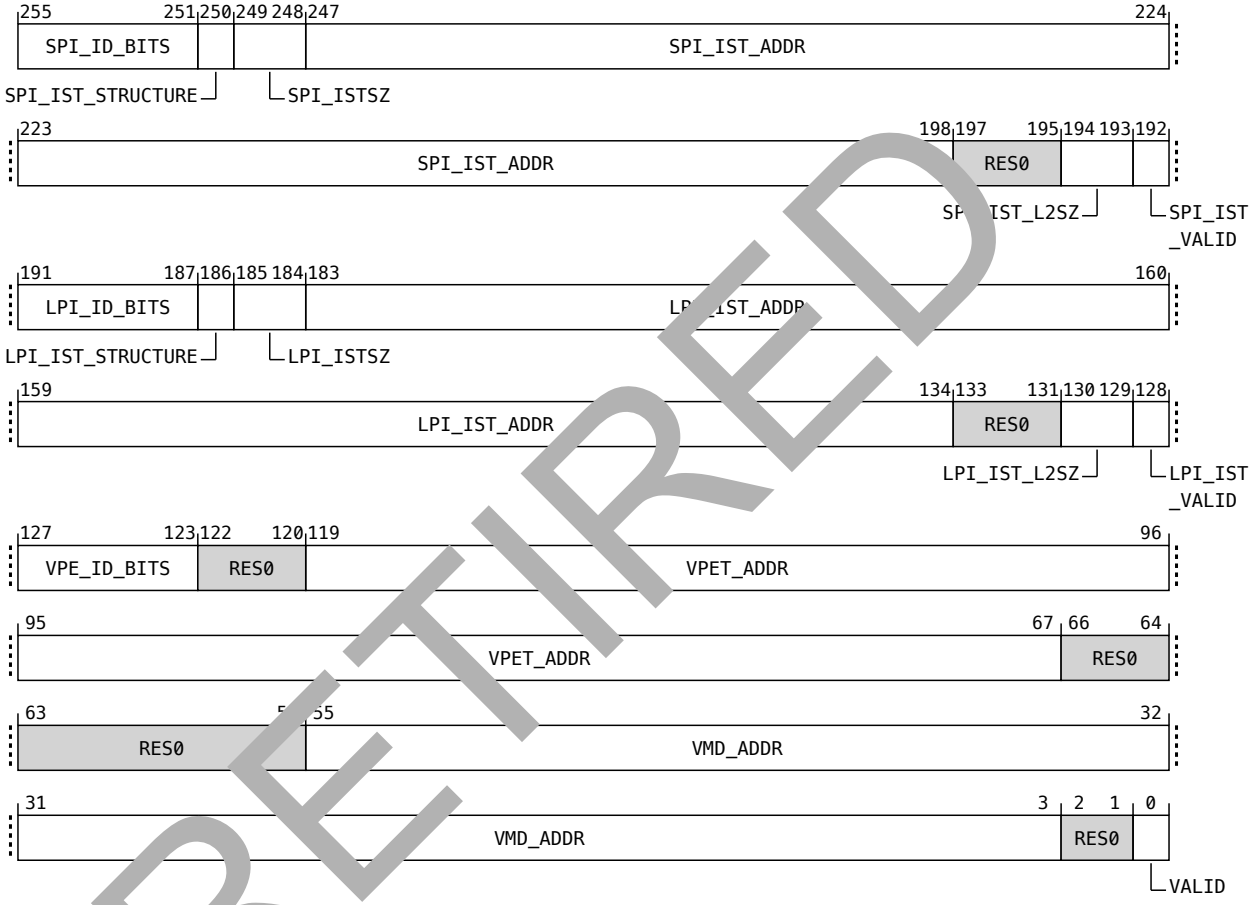
### 11.2.2 L2_VMTE, Level 2 VM table entry

The L2_VMTE characteristics are:

#### Attributes

L2_VMTE is a 32-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry describes an invalid VM. |
| 0b1 | This entry describes a valid VM. |

#### Bits [2:1]

Reserved, RES0.

#### VMD_ADDR, bits [55:3]

**When IRS_IDR3.VMD == 1:**

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Bits[55:3] of the base address of the VM Descriptor for the VM described by this entry.

The VM Descriptor address is aligned to its size as follows:

- Bits[N:0] of the resulting address are 0 where N = IRS_IDR3.VMD_SZ - 1.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

**Otherwise:**

Reserved, RES0.

### Bits [63:56]

Reserved, RES0.

### Bits [66:64]

Reserved, RES0.

### VPET_ADDR, bits [119:67]

Bits[55:3] of the base physical address of the array of VPE table entries for the VM described by this entry.

The VPE table always uses a linear structure and is aligned as follows:

- Bits[N:0] of the resulting address are 0 where N = 2 + VPE_ID_BITS.
- This means that the level 2 VPETE array is aligned to the size of the array.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

### Bits [122:120]

Reserved, RES0.

### VPE_ID_BITS, bits [127:123]

The number of VPE IDs which are supported for this VM.

The VPE table must contain (2 ^ VPE_ID_BITS) VPE table entries in total.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than read back the field.

The maximum value is reported in IRS_IDR4.VPE_ID_BITS.

### LPI_IST_VALID, bit [128]

LPI_IST_ADDR points to a valid IST

| LPI_IST_VALID | Meaning |
|---|---|
| 0b0 | LPI_IST_ADDR is not valid. All accesses to LPI configuration and state for the VM are dropped. |
| 0b1 | LPI_IST_ADDR is valid and points to a valid IST. |

### LPI_IST_L2SZ, bits [130:129]

Level 2 IST size when a 2-level IST structure is used.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| LPI_IST_L2SZ | Meaning |
|---|---|
| 0b00 | A level 2 IST is 4KB. |
| 0b01 | A level 2 IST is 16KB. |
| 0b10 | A level 2 IST is 64KB. |

Values not defined above are reserved.

IRS_IDR2.IST_L2SZ reports the supported values.

If LPI_ID_BITS <= (10 - LPI_ISTSZ) + (2 * LPI_IST_L2SZ) and LPI_IST_STRUCTURE is 1, all of the following are true:

- The IST consists of a single level 1 IST entry and a single level 2 IST.
- The level 2 IST contains (2 ^ LPI_ID_BITS) entries.
- The IRS is allowed to access the full level 2 IST size as specified by LPI_IST_L2SZ.

Arm recommends that LPI_IST_STRUCTURE is 0 when LPI_ID_BITS <= (10 - LPI_ISTSZ) + (2 * LPI_IST_L2SZ).

If programming a reserved value or an unsupported value, IRS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

When LPI_IST_STRUCTURE is 0, this field is RES0.

### Bits [133:131]

Reserved, RES0.

### LPI_IST_ADDR, bits [183:134]

Bits[55:6] of the base physical address of the LPI IST for the VM described by this entry.

When LPI_IST_STRUCTURE is 0, LPI_IST_ADDR points to a linear IST and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N = Max(5, ((LPI_ISTSZ + 2) + LPI_ID_BITS) - 1).

  This means that the level 2 IST entry array is aligned to the size of the array or to a 64-bytes boundary when its size is smaller than 64 bytes.

When LPI_IST_STRUCTURE is 1, LPI_IST_ADDR points to the level 1 table in a 2-level IST and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N depends on LPI_IST_L2SZ and LPI_ID_BITS as follows:

  N = Max(5, LPI_ID_BITS - ((10 - LPI_ISTSZ) + (2 * LPI_IST_L2SZ)) + 2)

- This means that the level 1 IST is aligned to the size of the level 1 IST entry array or to a 64-byte boundary when its size is smaller than 64 bytes.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

### LPI_ISTSZ, bits [185:184]

The size of each level 2 IST entry in the virtual LPI IST.

Values not defined above are reserved.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

If this field is programmed to specify a size smaller than the minimum required size or programmed to a reserved value, it is treated as having the value corresponding to the minimum required size for all other purposes than a read of the entry.

| LPI_ISTSZ | Meaning |
|---|---|
| 0b00 | The size of a level 2 LPI IST entry is 4 bytes. |
| 0b01 | The size of a level 2 LPI IST entry is 8 bytes. |
| 0b10 | The size of a level 2 LPI IST entry is [illegible] bytes. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

## LPI_IST_STRUCTURE, bit [186]

Structure of the LPI IST pointed to by LPI_IST_ADDR.

| LPI_IST_STRUCTURE | Meaning |
|---|---|
| 0b0 | A linear IST structure is used. |
| 0b1 | A 2-level IST structure is used. |

Values not defined above are reserved.

When programming a reserved value, the IRS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid value.

When IRS_IDR2.IST_LEVELS is 0, this field is treated as 0 for all other purposes than reading back the field.

## LPI_ID_BITS, bits [191:187]

The number of LPIs which are supported for this VM.

The IST must contain 2^(LPI_ID_BITS) level 2 IST entries in total.

The minimum value for this field is IRS_IDR2.MIN_LPI_ID_BITS.

If programmed to a value smaller than the minimum, the field is treated as having the minimum value for all other purposes than reading back the field.

The maximum value for this field is IRS_IDR2.ID_BITS.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field. The maximum value is reported in IRS_IDR2.ID_BITS.

## SPI_IST_VALID, bit [192]

Whether SPI_IST_ADDR points to a valid IST.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

| SPI_IST_VALID | Meaning |
|---|---|
| 0b0 | SPI_IST_ADDR is not valid. All accesses to SPI configuration and state for the VM are dropped. |
| 0b1 | SPI_IST_ADDR is valid and points to a valid IST. |

### SPI_IST_L2SZ, bits [194:193]

Level 2 IST size when a 2-level SPI IST structure is used.

| SPI_IST_L2SZ | Meaning |
|---|---|
| 0b00 | A level 2 IST is 4KB. |
| 0b01 | A level 2 IST is 16KB. |
| 0b10 | A level 2 IST is 64KB. |

Values not defined above are reserved.

IRS_IDR2.IST_L2SZ reports the supported values.

If SPI_ID_BITS <= ((10 - SPI_ISTSZ) + (2 * SPI_IST_L2SZ)) and SPI_IST_STRUCTURE is 1, all of the following are true:

- The IST consists of a single level 1 IST entry and a single level 2 IST.
- The level 2 IST contains (2 ^ SPI_ID_BITS) entries.
- The IRS is allowed to access the full level 2 IST size as specified by SPI_IST_L2SZ.

Arm recommends that SPI_IST_STRUCTURE is 0 when SPI_ID_BITS <= (10 - SPI_ISTSZ) + (2 * SPI_IST_L2SZ).

If programming a reserved value or an unsupported value, the IRS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid and supported value.

When SPI_IST_STRUCTURE is 0, this field is RES0.

### Bits [195]

Reserved, RES0.

### SPI_IST_ADDR, bits [247:198]

Bits[55:6] of the base physical address of the SPI IST for the VM described by this entry.

When SPI_IST_STRUCTURE is 0, SPI_IST_ADDR points to a linear IST and all of the following are true:

- Bits[N:0] of the resulting address are 0 where N = Max(5, ((SPI_ISTSZ + 2) + SPI_ID_BITS) - 1).
- This means that the level 2 IST entry array is aligned to the size of the array or to a 64-byte boundary when its size is smaller than 64 bytes.

When SPI_IST_STRUCTURE is 1, SPI_IST_ADDR points to the level 1 table in a 2-level IST and all of the following are true:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Bits[N:0] of the resulting address are 0 where N depends on the SPI_IST_L2SZ and SPI_ID_BITS field as follows:

  N = Max(5, SPI_ID_BITS - ((10 - SPI_ISTSZ) + (2 * SPI_IST_L2SZ)) + 2)

- This means that the level 1 IST is aligned to the size of the level 1 IST entry array or to a 64-byte boundary when its size is smaller than 64 bytes.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

### SPI_ISTSZ, bits [249:248]

The size of each level 2 IST entry in the virtual SPI IST.

Values not defined above are reserved.

If this field is programmed to specify a size smaller than the minimum required size or programmed to a reserved value, it is treated as having the value corresponding to the minimum required size for all other purposes than a read of the entry.

| SPI_ISTSZ | Meaning |
|---|---|
| 0b00 | The size of a level 2 SPI IST entry is 4 bytes. |
| 0b01 | The size of a level 2 SPI IST entry is 8 bytes. |
| 0b10 | The size of a level 2 SPI IST entry is 16 bytes. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

### SPI_IST_STRUCTURE, bit [250]

Structure of the SPI IST pointed to by SPI_IST_ADDR.

| SPI_IST_STRUCTURE | Meaning |
|---|---|
| 0b0 | A linear IST structure is used. |
| 0b1 | A 2-level IST structure is used. |

Values not defined above are reserved.

When programming a reserved value, the IRS behavior is CONSTRAINED UNPREDICTABLE to any behavior which could be achieved by programming a valid value.

When IRS_IDR2.IST_LEVELS is 0, this field is treated as 0 for all other purposes than reading back the field.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### SPI_ID_BITS, bits [255:251]

The number of SPIs which are supported for this VM.

The SPI IST must contain 2^(SPI_ID_BITS) level 2 IST entries in total.

The minimum value for this field is 0, which is equivalent to a minimum of 1 SPI in the VM.

If the VM has no SPIs, SPI_IST_VALID is 0.

If this field is programmed to a value larger than the maximum, it is treated as having the maximum value for all other purposes than reading back the field. The maximum value is reported in IRS_IDR2.ID_BITS.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

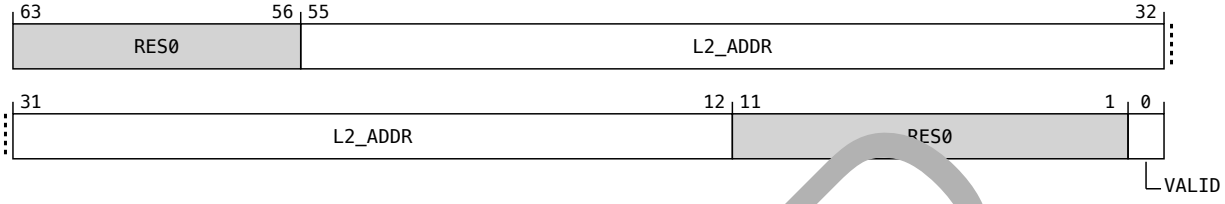
### 11.2.3 L1_ISTE, Level 1 interrupt state table entry

The L1_ISTE characteristics are:

#### Attributes

L1_ISTE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Whether the entry is valid.

| VALID | Meaning |
|---|---|
| 0b0 | This entry is invalid. All INTIDs covered by this entry are unreachable. |
| 0b1 | This entry is valid and L2_ADDR points to a level 2 IST. |

#### Bits [11:1]

Reserved, RES0.

#### L2_ADDR, bits [55:12]

Bits[55:12] of the address of the start of the array of level 2 IST entries.

Bits[N:0] of the resulting address are 0 where N = 11 + (2 * L2SZ)

This means that the level 2 IST array is aligned to the size of a level 2 IST.

The level 2 array is accessed using the same PAS as the level 1 ISTE.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

#### Bits [63:56]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 11.2.4 L2_ISTE, Level 2 interrupt state table entry

The L2_ISTE characteristics are:

#### Purpose

This data structure shows the 4 lowest address bytes of a single level 2 ISTE.

The size of a complete level 2 ISTE can be 4 bytes, 8 bytes, or 16 bytes.

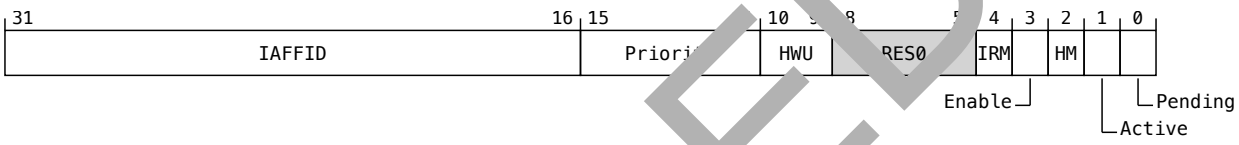
If the size of a level 2 ISTE is more than 4 bytes, the higher address bytes are RES0.

See *4.7 The interrupt state table (IST)* for more information.

#### Attributes

L2_ISTE is a 4-byte structure.

#### Field descriptions




#### Pending, bit [0]

Interrupt Pending state

| Pending | Meaning |
|---|---|
| 0b0 | The interrupt is not Pending. |
| 0b1 | The interrupt is Pending. |

#### Active, bit [1]

Interrupt [illegible] state

| Active | Meaning |
|---|---|
| 0b0 | The interrupt is not Active. |
| 0b1 | The interrupt is Active. |

#### HM, bit [2]

Handling mode of the interrupt.

| HM | Meaning |
|---|---|
| 0b0 | Edge: The interrupt becomes Pending when the interrupt is acknowledged. |
| 0b1 | Level: The interrupt Pending state is unmodified when the interrupts is acknowledged. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### Enable, bit [3]

Interrupt Enabled setting

| Enable | Meaning |
|---|---|
| 0b0 | The interrupt is Disabled. |
| 0b1 | The interrupt is Enabled. |

### IRM, bit [4]

Interrupt Routing mode.

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted. |
| 0b1 | The interrupt Routing mode is 1ofN. |

This field is RES0, if any of the following are true:

- IRS_IDR0.ONE_N is 0.
- The entry is part of a virtual IST and IRS_IDR0.VIRT_ONE_N is 0.

### Bits [8:5]

Reserved, RES0.

### HWU, bits [10:9]

Reserved for hardware use.

This field should be zero when an IST becomes valid and is otherwise IMPLEMENTATION DEFINED.

### Priority, bits [15:11]

The Priority value of the interrupt.

Bits [4:N] of the Priority value are implemented, where N = (4 - IRS_IDR1.PRI_BITS). Unimplemented bits are RES0.

This means that when fewer than 5 bits of priority is implemented, Priority[14 - IRS_IDR1.PRI_BITS:11] are RES0.

### IAFFID, bits [31:16]

Interrupt Affinity ID.

When this entry is part of a physical IST, this field specifies a PE.

When this entry is part of a virtual IST, this field specifies a VPE.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

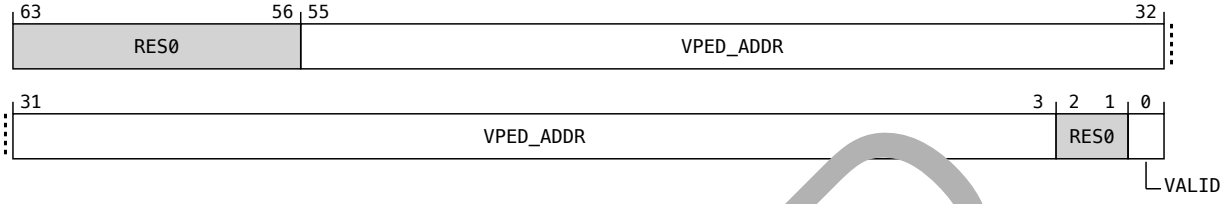
### 11.2.5 VPETE, VPE table entry

The VPETE characteristics are:

#### Attributes

VPETE is a 8-byte structure.

#### Field descriptions




#### VALID, bit [0]

Entry is valid

| VALID | Meaning |
|---|---|
| 0b0 | This entry describes an invalid VPE. |
| 0b1 | This entry describes a valid VPE. |

#### Bits [2:1]

Reserved, RES0.

#### VPED_ADDR, bits [55:3]

Bits[55:3] of the base address of the VPE Descriptor for the VPE described by this entry if used.

The VPE Descriptor address is aligned to its size as follows:

- Bits[N:0] of the resulting address are 0 where N = IRS_IDR4.VPED_SZ - 1.

In implementations that support fewer than 56 bits of physical address, any unimplemented upper bits are RES0. The number of implemented address bits is reported in IRS_IDR0.PA_RANGE.

#### Bits [63:56]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 11.2.6 VM_DESC, VM descriptor

The VM_DESC characteristics are:

#### Purpose

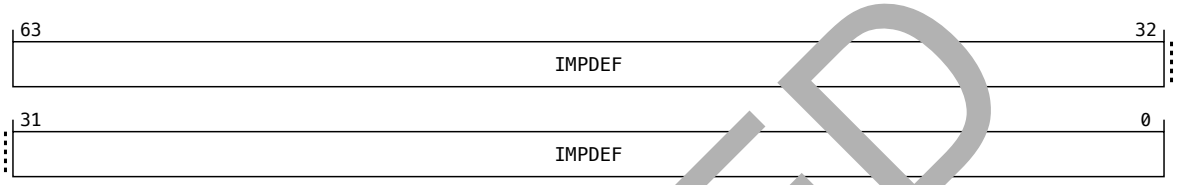
The size, format, and content of this structure is IMPLEMENTATION DEFINED.

This shows an example size of 8-byte.

#### Attributes

VM_DESC is a 8-byte structure.

#### Field descriptions




#### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### 11.2.7 VPE_DESC, VPE descriptor

The VPE_DESC characteristics are:

#### Purpose

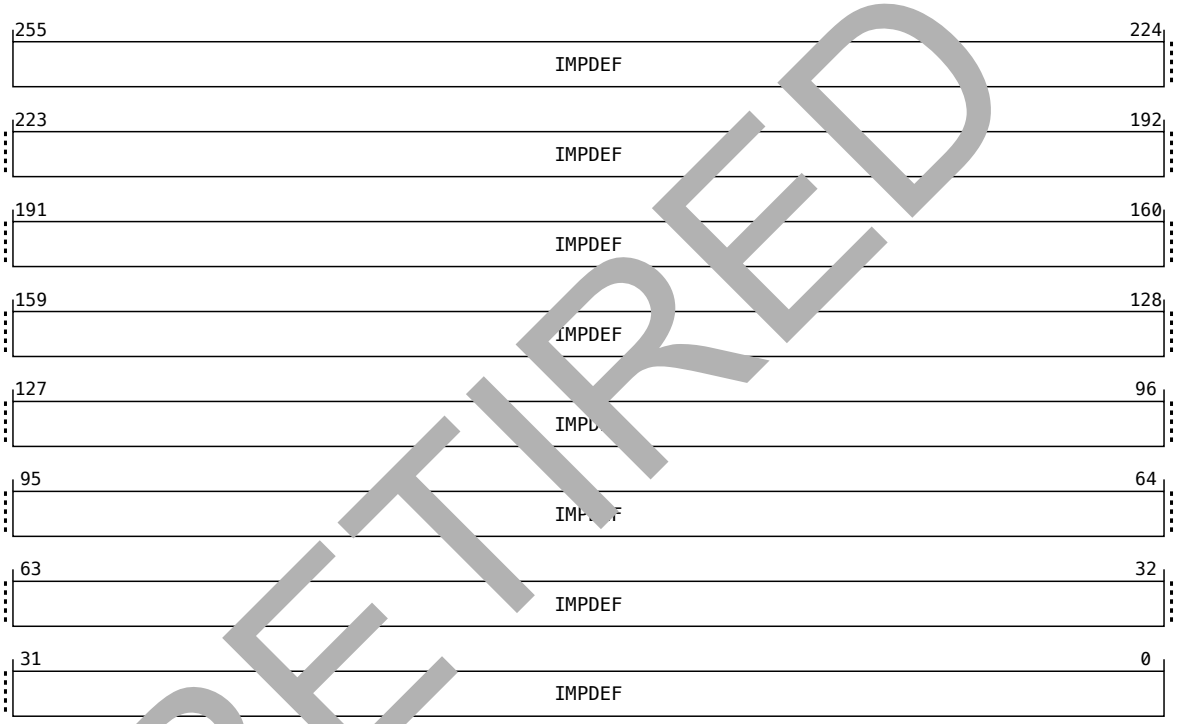
The size, format, and content of this structure is IMPLEMENTATION DEFINED.

This shows an example size of 32-byte.

#### Attributes

VPE_DESC is a 32-byte structure.

#### Field descriptions




#### IMPLEMENTATION DEFINED, bits [255:0]

IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Part A
## GRCv5 Stream Protocol interface

RETIRED

# Chapter A1
# GICv5 Stream Protocol overview

RETIRED

$I_{TGPMY}$ The GICv5 Stream Protocol is one approach to connecting an IRS and a CPU interface. The GICv5 Stream Protocol supports independent development of an IRS and a PE. Arm recommends that a GICv5 implementation uses the GICv5 Stream Protocol.

$I_{VPFYV}$ The GICv5 Stream Protocol is based on the GIC Stream Protocol defined by GICv3[3].

$D_{KNBQZ}$ An *end-point* means either a CPU interface or an interrupt source.

$I_{WYMBT}$ The GICv5 Stream Protocol supports the connection between an IRS and one or more end-points.

$D_{NXTBL}$ The connection between an IRS and a CPU interface is referred to as an *Interrupt Handling channel*.

$D_{SBHHG}$ The connection between an IRS and an interrupt source is referred to as an *Interrupt Signaling channel*.

$I_{FSTMD}$ GICv5 Stream Protocol uses a transport layer formed of a pair of AMBA AXI5-Stream Protocol links:

- From the IRS to one or more end-points.
- From one or more end-points to the IRS.

See *AMBA® AXI Protocol Specification*[6] for information about the AMBA AXI5-Stream Protocol.

$D_{WBVHL}$ *AXI Stream connection* refers to the transport layer connection between an IRS and one or more end-points.

$D_{DCXRX}$ A command sent by the IRS to an end-point is defined as a *Downstream* command.

$D_{PTTTR}$ A command sent by an end-point to the IRS is defined as an *Upstream* command.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter A2
# AMBA AXI5-Stream Transport Layer

RETIRED

IVSYKR The GICv5 Stream Protocol is based on the following unidirectional AXI5-Stream connections:

- A downstream AXI5-Stream Interface containing connections from an IRS to one or more end-points.
- An upstream AXI5-Stream Interface containing connections from one or more end-points to an IRS.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

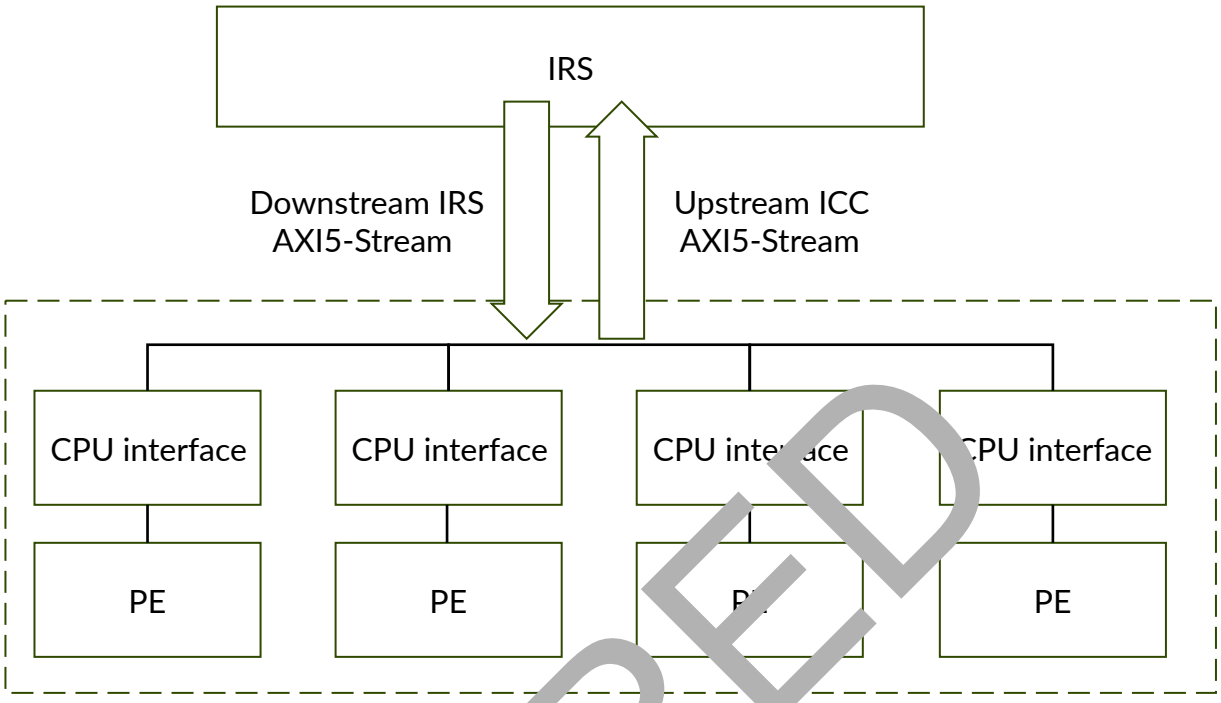



**Figure A2.1: GICv5 Stream Protocol Interface**

A GICv5 Stream Protocol connection connects a combination of end-point types:

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

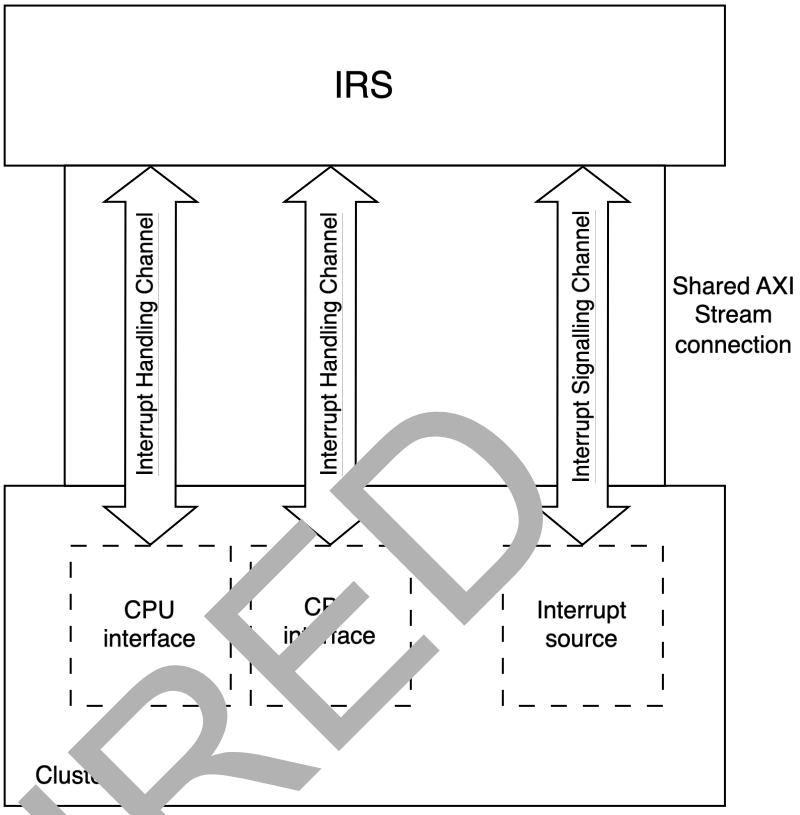



**Figure A2.2: GICv5 Stream Protocol Interface**

An IRS might have multiple GICv5 Stream Protocol connections to different groups of PEs, for example to connect to multiple clusters.

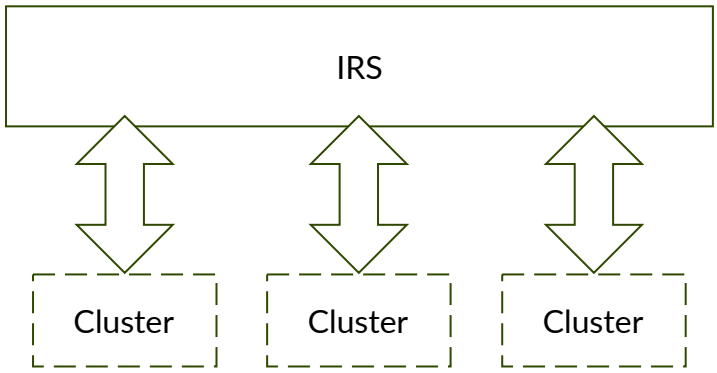



**Figure A2.3: IRS with multiple GICv5 Stream Interfaces**

$R_{XRVGD}$ An interconnect between an IRS and a CPU interface must ensure that the stream packet sequence is transferred over the stream protocol interface in the same order in which it was created.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## A2.1 Signals

$R_{JSSRX}$ The interface requires a global clock, ACLK, and a reset signal, ARESETn.

$I_{VPLGV}$ For the GICv5 Stream Protocol, each stream interface is identified by a prefix to the AXI Stream signal names:

- Downstream signals from an IRS to the CPU interface are prefixed with the letters IRS.
- Upstream signals from the CPU interface to an IRS are prefixed with the letters ICC.

$R_{GQLTR}$ The following table illustrates the GICv5 Stream Protocol interface signals from the IRS to the downstream end-point:

| Signal | Description |
|---|---|
| IRSTVALID | When set to 1, this signal indicates that the Requester is driving a valid transfer. |
| IRSTREADY | When set to 1, this signal indicates that the Completer can accept a transfer in the current cycle. |
| IRSTDATA[BN:0] | The interface data path. |
| IRSTLAST | When set to 1, this signal indicates the final transfer of a command. |
| IRSTID | This signal identifies the channel the transfer is associated with. |
| IRSTDEST[N:0] | This signal identifies the target CPU interface to provide routing information for the stream. |
| IRSTWAKEUP | Indicates if there is activity associated with the AXI5-Stream interface. |

The following table illustrates the GICv5 Stream Protocol interface signals from the end-point to the upstream IRS:

| Signal | Description |
|---|---|
| ICCTVALID | When set to 1, this signal indicates that the Requester is driving a valid transfer. |
| ICCTREADY | When set to 1, this signal indicates that the Completer can accept a transfer in the current cycle. |
| ICCTDATA[BN:0] | The interface data path. |
| ICCTLAST | When set to 1, this signal indicates the final transfer of a command. |
| ICCTID[N:0] | This signal identifies the originating CPU interface, to provide routing information for the stream. |
| ICCTDEST | This signal identifies the channel the transfer is associated with. |
| ICCTWAKEUP | Indicates if there is activity associated with the AXI5-Stream interface. |

Where:

- BN is the number associated with the most significant bit on a datapath that is required to be an integral number of bytes wide.
- N is the value $\log_2(M)$ rounded up to the nearest integer, where M is the number of PEs supported by the interface plus 1.

$I_{SRVVS}$ In the GICv5 Stream Protocol, all commands have a length which is a multiple of 16 bits.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{DWDQB}$ The xTKEEP and xTSTRB signals are defined as optional in *AMBA® AXI-Stream Protocol Specification*[12].

If the AXI5-Stream interfaces used for a GICv5 Stream Protocol link have a data path that is 16-bit or smaller, Arm recommends that xTKEEP is not implemented. Otherwise, Arm recommends that xTKEEP is implemented.

If the xTKEEP signal is implemented, it is asserted for at least the first byte required for the command. The sender may de-assert xTKEEP for other bytes, if all remaining bytes in the command are `0b00`. Any byte for which xTKEEP is de-asserted is treated as being `0b00` by the receiver.

If the xTKEEP signal is not implemented, any bytes not required to transmit a command are set to `0b00`.

Arm recommends that the xTSTRB signal is not implemented. If xTSTRB is implemented, it is asserted for all bytes for which xTKEEP is asserted and deasserted for all the other bytes.

$R_{ZZDQV}$ The xTLAST signal is asserted to indicate the last transfer for a command.

A sender may assert xTLAST before all bytes of a command have been transferred. Any untransferred bytes in the command are treated as having the value `0b00` by the receiver.

A sender may assert xTLAST late, making additional transfers not needed for the command. The receiver ignores any bytes not needed to represent the command.

$I_{RLKJV}$ If xTLAST is asserted late and xTKEEP is implemented, for the additional transfers, xTKEEP is deasserted.

$I_{QDLHQ}$ If parity checking is required, Arm recommends implementing the support for parity in AXI Stream defined in *AMBA® AXI-Stream Protocol Specification*[12].

$I_{TNVVV}$ For further information about the signals used by the GICv5 Stream Protocol interface, and for details about handshaking, see *AMBA® AXI-Stream Protocol Specification*[12].

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## A2.2 Channel identification

$I_{NMPSG}$ The AXI Stream connection can connect one IRS to multiple end-points. A connection between the IRS and an end-point is referred to as a channel. An Interrupt Handling channel connects an IRS to a CPU interface. An Interrupt Signaling channel connects an IRS to an interrupt source.

$I_{PDQSW}$ The xTID and xTDEST signals are used to route packets and identify the type of channel.

$R_{BRZBR}$ For downstream commands, IRSTID indicates the channel type:

**0: Interrupt Handling channel** IRSTDEST indicates the target CPU interface. Values are allocated contiguously from 0, in order of ascending affinity.
**1: Interrupt Signaling channel** IRSTDEST indicates the target interrupt source.

$R_{RBSPN}$ For upstream commands, ICCTDEST indicates the channel type:

**0: Interrupt Handling channel.** ICCTID indicates the originating CPU interface.
**1: Interrupt Signaling channel.** ICCTID indicates the originating interrupt source.

$R_{TCSCP}$ The order and encoding of end-points is the same for IRSTDEST and ICCTID.

$R_{JCRRQ}$ The values used to identify CPU interfaces are allocated contiguously from 0, in order of ascending affinity.

$R_{WZJVS}$ The values used to identify interrupt sources is IMPLEMENTATION DEFINED.

$I_{ZJTKX}$ Arm recommends that the values used to identify interrupt sources are allocated contiguously from 0.

$R_{NDDWL}$ Transfers on the same AXI5-Stream Interface for different commands are not interleaved. Once transmission of a command is started subsequent transfers contain the same command, with the same values of xTID and xTDEST, until xTLAST is asserted.

$I_{RQBRS}$ The restriction on interleaving transfers does not prevent upstream and downstream commands being sent in parallel.

$I_{DMDNC}$ An example system might have two CPU interfaces and an interrupt source sharing an AXI Stream connection, as shown in the diagram below:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

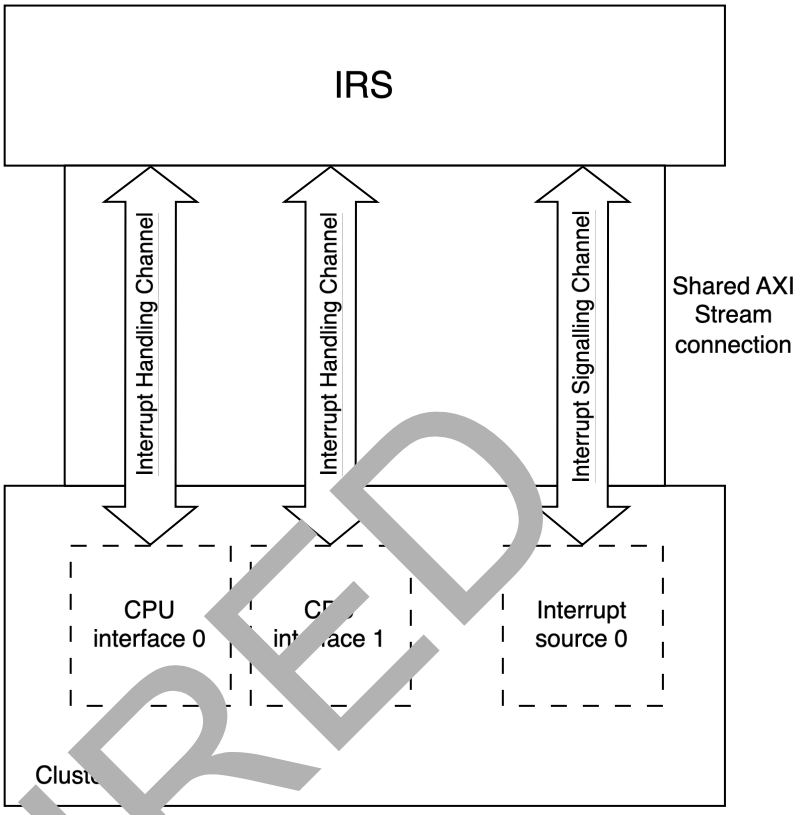



**Figure A2.4: GICv5 Stream Protocol Interface**

Downstream commands:

| IRSTID | IRSTDEST | Channel type | Target |
|---|---|---|---|
| 0 | 0 | Interrupt Handling | CPU interface 0 |
| 0 | 1 | Interrupt Handling | CPU interface 1 |
| 1 | 0 | Interrupt Signaling | Interrupt source 0 |

Upstream commands:

| ICCTDEST | ICCTID | Channel type | Source |
|---|---|---|---|
| 0 | 0 | Interrupt Handling | CPU interface 0 |
| 0 | 1 | Interrupt Handling | CPU interface 1 |
| 1 | 0 | Interrupt Signaling | Interrupt source 0 |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A2.3 Link status

$D_{WDBCG}$ When the AXI Stream connection is described as *online*, data can be transferred across the upstream and downstream AXI Stream links. When a AXI Stream connection is described as *offline*, no data can be transferred across the AXI Stream links.

$R_{SMLNR}$ When one or more of the channels that share the AXI Stream connection is online the AXI Stream connection is online.

$R_{DGDFT}$ When none of the channels that share the AXI Stream connection is online, it is IMPLEMENTATION DEFINED whether the AXI Stream connection is online or offline.

$R_{NBCWM}$ How the AXI Stream connection transitions from online to offline, or offline to online, is IMPLEMENTATION DEFINED.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter A3
# **Common behaviors**

RETIRED

$I_{QHRMH}$ This section defines behaviors common to the Interrupt Handling and Interrupt Signaling channels.

$D_{PWLWC}$ Some commands require an acknowledgement from the receiver. A command that requires an acknowledgement is defined as *outstanding* until the acknowledgement is received.

$D_{TYKBT}$ A command that requires acknowledgement is described as *complete* once the acknowledgement is received. All other commands are described as complete once received by the recipient.

$R_{NQQBH}$ If multiple transmitters are required to transmit a command, the receiver does not acknowledge the command until all of the command has been received.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter A4
# Interrupt Handling channel

RETIRED

$I_{HTHMK}$ The Interrupt Handling channel defines the connection between one CPU interface and one IRS, it includes the following capabilities:

- Forwarding candidate HPPIs to the CPU interface.
- Activating and deactivating interrupts as they are handled.
- Managing the virtual interrupt resident VPE.
- Managing interrupt configuration and state.

$I_{LYNNC}$ The rules in this section only apply to the Interrupt Handling channel.


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## A4.1 Command summary

$I_{LHGVT}$ The upstream commands that a CPU interface can send to the IRS are:

| ID | Command | Description |
|---|---|---|
| 0x0 | UpstreamControl | Communicates control information from the CPU interface to the IRS. |
| 0x1 | - | - |
| 0x2 | Activate | Activates a previously sent interrupt. |
| 0x3 | Release | Releases a previously sent interrupt. |
| 0x4 | DownstreamControl Ack | Acknowledges a DownstreamControl comma |
| 0x5 | - | - |
| 0x6 | Deactivate | Deactivates an interrupt. |
| 0x7 | IMPLEMENTATION DEFINED | Reserved for IMPLEMENTATION EFINED functi lit |
| 0x8 | SetEnabled | Sets the individual Enabled va of an terrupt. |
| 0x9 | SetTarget | Sets an LPI or SPI's tar E. |
| 0xA | SetPriority | Sets an interrupt's ority. |
| 0xB | SetHandling | Sets an interrupt's ndli |
| 0xC | RequestConfig | Requests an rrupt's nfiguration and state |
| 0xD | Sync | Requ s confirma n that effects of previous commands are globally observable. |
| 0xE | SetPending | S s a PI or SPI's P ding state. |
| 0xF | SetResident | Sets the cu nt resident VPE. |

$I_{PYPQZ}$ The downstream mma s that an IRS can send to the CPU interface are:

| ID | Command | es ption |
|---|---|---|
| 0x0 | UpstreamC rol A | Acknowledges an UpstreamControl command. |
| 0x1 | Forward | Sets an interrupt as pending on the CPU interface. |
| 0x2 | Activate Ack | Acknowledges an Activate command. |
| 0x3 | Recall | Recalls a previously sent pending interrupt. |
| 0x4 | DownstreamControl | Communicates control information from the IRS to the CPU interface. |
| 0x5 | WakeRequest | Requests the channel is brought online. |
| 0x6 | Deactivate Ack | Acknowledges a Deactivate command. |
| 0x7 | IMPLEMENTATION DEFINED | Reserved for IMPLEMENTATION DEFINED functionality. |
| 0x8 | Set Ack | Acknowledges a SetEnabled, SetPending, SetPriority, SetTarget, or SetHandling command. |
| 0x9 | - | - |
| 0xA | - | - |
| 0xB | - | - |

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| ID | Command | Description |
|---|---|---|
| 0xC | RequestConfig Ack | Acknowledges a RequestConfig command and returns the requested data. |
| 0xD | Sync Ack | Acknowledges a Sync command, indicating that the effect of previous commands are globally observable. |
| 0xE | - | - |
| 0xF | SetResident Ack | Acknowledges a SetResident command. |

$R_{LDZYS}$ ID `0x7` in upstream and downstream is reserved for IMPLEMENTATION DEFINED functionality.

When supported, IMPLEMENTATION DEFINED commands do not impact the functionality of other commands. If not supported, IMPLEMENTATION DEFINED commands are ignored.

Arm strongly recommends that IMPLEMENTATION DEFINED commands are not used unless it can be guaranteed that both the sender and receiver agree on the usage.

$I_{LTNZK}$ Where a command operates on a specific Interrupt Domain, the `Domain` field indicates the target Domain.

Where a command can apply to a physical Interrupt Domain or a Virtual Interrupt Domain, the `Virtual` field indicates whether the command is physical or virtual.

An interrupt that targets the EL3 Domain cannot be virtual, as GIC does not support a virtual Domain in EL3.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.2 Outstanding commands

$R_{DGPBP}$ Commands which require acknowledgement are acknowledged in finite time, apart from Forward commands.

$R_{MHMBB}$ If a Forward is replaced by a subsequent Forward for the same Interrupt Domain, the CPU interface issues a Release or Activate for the replaced Forward in finite time.

$I_{YYGKH}$ The following commands require acknowledgement:

| Command | Direction | Acknowledged by |
|---|---|---|
| Activate | Upstream | Activate Ack or Forward with `ActivateAck` set to 1. |
| Deactivate | Upstream | Deactivate Ack |
| DownstreamControl | Downstream | DownstreamControl Ack. |
| RequestConfig | Upstream | RequestConfig Ack |
| Forward | Downstream | Activate or Release |
| | | SetResident with `Valid` set to 0. |
| | | UpstreamControl with Identifier set to `0b0001` (Quiesce) |
| SetEnabled | Upstream | Set Ack |
| SetPending | Upstream | Set Ack |
| SetHandling | Upstream | Set Ack |
| SetPriority | Upstream | Set Ack |
| SetResident | Upstream | SetResident Ack |
| SetTarget | Upstream | Set Ack |
| Sync | Upstream | Sync Ack |
| UpstreamControl | Upstream | UpstreamControl Ack |

All other commands do not require acknowledgement.

$I_{CWTZR}$ For commands that require acknowledgement, the architecture places limits on the number of outstanding commands that are permitted.

$I_{YYPPQ}$ An UpstreamControl command with Identifier set to `0b0000` (Reset) initiates a reset of the channel. Resetting the channel returns pending interrupts to the IRS and cancels outstanding commands. Acknowledges to commands issued before the reset started might still be received during the reset process.

$R_{YDBHS}$ For each Interrupt Handling channel, the IRS is permitted to have one DownstreamControl command outstanding.

$I_{KGXBV}$ The WakeRequest command does not have an acknowledge, meaning that is considered complete after being sent rather than requiring an explicit acknowledge. WakeRequests are implicitly acknowledged by the CPU interface sending UpstreamControl with Identifier set to `0b0000` (Reset) to bring the channel online.

$I_{MJDMH}$ Arm recommends that once an IRS has issued a WakeRequest command it does not send further wake requests until the next time the channel is offline.

$R_{PSRJH}$ For each Interrupt Handling channel, the CPU interface is permitted to have the following numbers of outstanding commands:

- One Activate command.
- One RequestConfig command.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- One SetResident command.
- One UpstreamControl command.
- One Sync command.
- One Deactivate command.
- In total, one of the following commands:
  - SetEnabled.
  - SetHandling.
  - SetPending.
  - SetPriority.
  - SetTarget.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.3 Connection management

$D_{GRHNS}$ When an Interrupt Handling channel is described as *online*, upstream and downstream commands can be sent between the IRS and CPU interface. When an Interrupt Handling channel is described as *offline*, the only commands that can be sent are those to bring the channel online.

$R_{GYWDK}$ When an Interrupt Handling channel is offline:

- The IRS sends no commands other than WakeRequest.
- The CPU interface sends no commands other than UpstreamControl with Identifier set to `0b0000` (Reset).

$R_{WZVTW}$ If an Interrupt Handling channel is offline, the IRS may send a WakeRequest command to request the channel to be brought online.

Whether an IRS sends wake requests over GICv5 Stream is IMPLEMENTATION DEFINED and an IRS might use other IMPLEMENTATION DEFINED mechanisms as well as or instead of GICv5 Stream.

$I_{TXLVK}$ Arm expects an IRS to only send a wake request if there is a candidate HPPI for the P

$R_{ZPMWQ}$ The IRS does not send WakeRequest commands if the Interrupt Handling channel is online.

$I_{TSRMX}$ A WakeRequest is implicitly acknowledged by an UpstreamControl command with Identifier set to `0b0000` (Reset).

$R_{BTJBS}$ To bring an Interrupt Handling channel online, or reset an online channel:

- The CPU interface sends an UpstreamControl command with Identifier set to `0b0000` (Reset).
- One or more instances of:
  - The IRS sends a DownstreamControl command with Identifier set to `0b0001` (Flush).
  - The CPU interface acknowledges the DownstreamControl command, with Flush set to `1`.
- The IRS acknowledges the UpstreamControl command.

Once the UpstreamControl command is acknowledged, the Interrupt Handling channel is online.

$R_{RNZYL}$ When an UpstreamControl command with Identifier set to `0b0000` (Reset) is acknowledged:

- There are no outstanding upstream or downstream commands.
- There are no pending physical or virtual interrupts from the IRS pending on the CPU interface.
- There is no resident VPE.
- There are no 1ofN selection hints set.

$R_{MSHYJ}$ Between sending an UpstreamControl command with Identifier set to `0b0000` (Reset) and the command being acknowledged, the CPU interface sends no commands other than to acknowledge a DownstreamControl with Identifier set to `0b0001` (Flush).

When the IRS receives an UpstreamControl command with Identifier set to `0b0000` (Reset), within finite time the IRS will respond with a DownstreamControl with Identifier set to `0b0001` (Flush). Once it has sent the DownstreamControl command, the IRS sends no other commands until is has received DownstreamControl Ack and sent the UpstreamControl Ack.

$I_{FSWFW}$ The IRS might issue some other commands between receiving the UpstreamControl command with Identifier set to `0b0000` (Reset) and sending the DownstreamControl with Identifier set to `0b0001` (Flush). The CPU interface ignores any downstream commands it receives between sending the UpstreamControl command with Identifier set to `0b0000` (Reset) and receiving a DownstreamControl with Identifier set to `0b0001` (Flush).

$I_{PQTNY}$ When the CPU interface sends UpstreamControl command with Identifier set to `0b0000` (Reset) it cancels outstanding commands. If the channel is reset while online, it is possible due to transmission delays that some commands are sent before the reset but received after the reset. This can lead, for example, to acknowledges being received for commands that are no longer considered outstanding. The multi-stage reset sequence is intended to deal with this possibility.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{DWCKW}$ When the IRS sends a DownstreamControl command with Identifier set to `0b0001` (Flush), the Sequence field is set to a value based on an IMPLEMENTATION DEFINED scheme.

When the CPU interface acknowledges a Flush, it sets the Sequence field in DownstreamControl Acknowledge to the same value as provided in the Flush being acknowledged.

$R_{DLBTV}$ The IRS's response to receiving a DownstreamControl Ack command with Flush set to 1 depends on the received Sequence number and whether there is an outstanding DownstreamControl command with Identifier set to `0b0001` (Flush).

The acknowledge is ignored if either of the following is true:

- There is no outstanding DownstreamControl command with Identifier set to `0b0001` (Flush).
- The received Sequence number does not match that sent in the most recent DownstreamControl command with Identifier set to `0b0001` (Flush).

Otherwise, if none of the above are true, the IRS does one of:

- Send a DownstreamControl command with Identifier set to `0b0001` (Flush).
- Send an acknowledge to the UpstreamControl command completing the reset sequence.

$R_{BYZBL}$ If the CPU interface receives multiple DownstreamControl commands with Identifier set to `0b0001` (Flush), it is IMPLEMENTATION DEFINED whether it acknowledges each command or only the most recent. If the CPU interface acknowledges each command, it acknowledges them in the order they were received.

$I_{GSQPK}$ GICv5 uses IAFFIDs to identify the different connected PEs. The IRS informs the CPU interface of its IAFFID as part of bringing the Interrupt Handling Channel online.

Some systems might require an IMPLEMENTATION DEFINED sequence before the correct IAFFIDs are known for the CPU interfaces. If the Interrupt Handling Channel is brought online before this sequence is completed the IRS can re-communicate the correct IAFFID using a DownstreamControl command with Identifier set to `0b0010` (IAFFID).

In the PE, the IAFFID is represented in ICC_IAFFID_EL1.

$I_{CDBHF}$ If a system requires an IMPLEMENTATION DEFINED sequence to initialise the correct IAFFIDs, Arm recommends that this occurs no more than once between GIC resets.

$R_{GYHSP}$ The CPU interface does not cache the IAFFID value across resets of the Interrupt Handling Channel.

$I_{SYLKT}$ Under normal operation, an UpstreamControl command with Identifier set to `0b0000` (Reset) is sent only to bring the connection between the CPU interface and IRS online. In which case there are no other outstanding commands, outstanding pending interrupts or resident VPE.

However, the CPU interface may send an UpstreamControl command with Identifier set to `0b0000` when the Interrupt Handling channel is already online. Sending the command resets the connection, returning all pending interrupts and cancelling outstanding commands.

When an online connection is being reset, it is possible that the CPU interface receives downstream commands between sending the UpstreamControl and seeing the acknowledgement. The CPU interface does not respond to these commands and the IRS does not expect to receive responses.

$R_{HXGXY}$ To take an Interrupt Handling channel offline:

- The CPU interface sends an UpstreamControl command with Identifier set to `0b0001` (Quiesce).
  - The CPU interface sends no further commands, other than acknowledgements to downstream commands, until UpstreamControl Ack is received.
- The IRS completes any outstanding requests from the CPU interface, then sends an UpstreamControl Ack.

Once the UpstreamControl command is acknowledged, the Interrupt Handling channel is offline.

$I_{QDJXY}$ There is a single Interrupt Handling channel between the CPU interface and the IRS, shared by all implemented Interrupt Domains. Therefore, the UpstreamControl commands are independent of Interrupt Domain.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$R_{YZFJG}$ When the Interrupt Handling channel is offline, the IRS and CPU interface treat all previously forwarded physical or virtual interrupts having been recalled.

$I_{QNSGW}$ Taking the Interrupt Handling channel offline acts as an implicit Release for any pending physical or virtual interrupts of any Interrupt Domain, including Forwards received after the Quiesce is sent. If there is an outstanding acknowledge for an interrupt, the CPU interface must send the Activate command before sending the UpstreamControl command with Identifier set to `0b0001` (Quiesce).

$I_{GWCSK}$ The UpstreamControl command provides a mechanism for the CPU interface to provide hints to the 1ofN selection algorithm in the IRS. The algorithm for 1ofN selection is IMPLEMENTATION DEFINED and might ignore any hints provided by the CPU interface. Whether a CPU interface sends 1ofN hints is IMPLEMENTATION DEFINED.

$R_{FMGVS}$ When an Interrupt Handling channel is online, the CPU interface can set hints for 1ofN target selection.

The CPU interface sends an UpstreamControl command with:

- Identifier set to `0b0010` to set 1ofN hints.
- Identifier set to `0b0011` to clear 1ofN hints.

The Data field in the UpstreamControl command specifies which hints are being set or cleared. The state of a 1ofN hint not specified in Data is unaffected by set or clear operations.

Unsupported or unknown hints are ignored by the IRS.

$I_{MLPDY}$ The CPU interface is permitted to clear hints that it has not previously set.

$I_{YCCYG}$ Each hint defines which Interrupt Domain, or Domains, it applies.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.4 Managing the resident VPE

$I_{XNGZK}$ When an Interrupt Handling channel transitions from offline to online, there is no resident VPE.

$R_{TPKML}$ To make a VPE resident, the CPU interface sends a SetResident command with `Valid` set to 1. The VPE being made resident is identified by the following fields:

- `Domain` - the Physical Interrupt Domain the VPE is associated with.
- `VM` - the virtual machine the VPE is associated with.
- `VPE` - the VPE within the specified virtual machine.

The IRS acknowledges the SetResident command with a SetResident Ack command with `Fault` set to indicate whether specified VPE was made resident.

The IRS does not generate a SetResident Ack until one of the following is t

- The specified VPE was made resident and one of the following i ue:
  - The IRS has forwarded the first virtual interrupt for the VP
  - The IRS has determined that there is no virtual interru to fo rd.
- The IRS has determined that it is unable to make the sp ied VPE re dent.

Once the SetResident Ack command with `Fault` set to 0 sent, the pecified V E is resident.

$I_{QPKWV}$ During the sequence for making a VPE resident, the IRS sen most one orward command for the VPE. If an alternative candidate HPPI for the VPE becomes ani le, the S for rds that interrupt after the SetResident Ack is sent.

$R_{JJTKF}$ During the sequence for making a VPE reside if th Forward command for the VPE it commits to sending SetResident Ack with F set to

$I_{RRBVK}$ If the IRS sends a Forward for the PE bei made r dent, it must acknowledge the SetResident with F set to 0. Therefore, the IRS must determ e that it can s cessfu y make the VPE resident before it sends the Forward.

$I_{HSHXS}$ An IRS might be unable to ake the quested VPE resident, for example because the VPE is invalid or the VPE is already resident on a di rent CPU in face.

$I_{VSJLW}$ There might be no tual interrupt to forw d because there is no candidate HPPI for the VPE.

$R_{HLJNT}$ To clear the resi t VPE ne CPU interface sends a SetResident command with `Valid` set to 0. When the IRS acknowledges the S sident co and by sending a SetResident Ack, there is no resident VPE.

When t IRS a nowled s t SetResident command by sending a SetResident Ack, the IRS and CPU interface trea l previo y forward virtual interrupts for the VPE as having been recalled.

$I_{QTCTP}$ Clear the resident VPE acts as an implicit Release for any outstanding pending virtual interrupts, returning the interrupts the IRS.

If there is any utstanding Activate for a virtual interrupt, the CPU interface must send the Activate command before sending SetResident.

$I_{LDFMM}$ The CPU interface might send a SetResident command with `Valid` set to 0 while earlier virtual commands remain outstanding. The IRS is not required to delay sending SetResident Ack until it has acknowledged those earlier virtual commands.

If the IRS returns SetResident Ack while earlier virtual commands are outstanding, the GIC IRI must ensure that ordering and barrier operation for the VPE are maintained, if the VPE is made resident on a different physical PE.

For example, consider the following sequence:

1. Physical PE0: VPE0 made resident.
2. Physical PE0: VPE0 sets priority of vINTID X to 0.
3. Physical PE0: VPE0 disables vINTID Y.
4. Physical PE0: VPE0 made not resident.
5. Physical PE1: VPE0 made resident.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

6. Physical PE1: VPE0 sets priority of vINTID X to 1.
7. Physical PE1: VPE0 issues a `GSB SYS`.

If, when VPE0 is made non-resident, the IRS returns SetResident Ack while virtual commands are outstanding, the GIC IRI would need to do all of the following:

- Detect that the SetPriority for VPE0 issued by physical PE1 is Ordered-after the SetPriority issued by phyiscal PE0.
- Detect that the SYNC issued for the GSB SYS on physical PE1 must wait on the SetEnabled issued by physical PE0.

$R_{VCZPC}$ The CPU interface does not send a SetResident command with `Valid` set to 1 when there is a resident VPE.

$R_{RZRYM}$ Between sending SetResident and receiving SetResident Ack, the CPU interface sends no commands with `Virtual` set to 1 other than SetPending, and commands that acknowledge downstream commands.

### A4.4.1 Interrupt Handling Channel behaviors when there is a resident VPE

$R_{MGZZN}$ All downstream and upstream commands with `Virtual` set to 1 other than SetPending are for the resident VPE. The VM targeted by SetPending is specified as part of the command's payload.

$R_{PYWKM}$ If the IRS receives a command with `Virtual` set to 1 where `Domain` does not correspond to the IRS's record of the resident VPE, the behavior depends on the type of the command as follows:

- RequestConfig Ack commands return `F` as 1 and zeros for all configuration fields.
- SetResident commands with `Valid` set to 0 clear the resident VPE, and `DB` is treated as being 0.
- All other commands are acknowledged but otherwise have no effect.

$R_{QMVYG}$ If the CPU interface receives a command with `Virtual` set to 1 and where `Domain` does not correspond to the current value of SCR_EL3.{NSE,NS}, the behavior depends on the type of the command as follows:

- Forward command: The command replaces any previously received virtual interrupts, but the new interrupt is not considered when determining the virtual HPPI.
  - If the Forward command results in an implicit Recall, the CPU interface sets `Domain` in the Release or Activate command to the Interrupt Domain of the interrupt that is being Recalled.
- Recall command: Any resulting Activate or Release command is issued with `Domain` set to the Interrupt Domain of the interrupt that is being Recalled.

### A4.4.2 Interrupt Handling Channel behaviors when there is no resident VPE

$R_{JRPVW}$ If there is no resident VPE:

- The IRI:
  - Processes SetResident commands with `Virtual` set to 1.
  - Processes SetPending commands with `Virtual` set to 1.
  - Does not issue Forward commands with `Virtual` set to 1, except as part of the sequence to make a VPE resident.
  - Acknowledges, but otherwise ignores, RequestConfig commands that have `Virtual` set to 1 with RequestConfig Ack commands that have `F` set to 1.
  - Acknowledges, but otherwise ignores, all other upstream commands with `Virtual` set to 1.
- The CPU interface:
  - Does not issue upstream commands with `Virtual` set to 1, other than SetPending and SetResident with `Valid` set to 1.
  - Downstream Forward commands with `Virtual` set to 1 are not considered when determining the virtual HPPI until a VPE is resident.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.5 Forwarding, recalling, and releasing interrupts

$I_{GLBSF}$ The IRS forwards a candidate HPPI for an Interrupt Domain, or for the resident VPE, using a Forward command. An interrupt can be retrieved from the CPU interface using a Recall command, or by sending another Forward command for the same Interrupt Domain.

For each Physical Interrupt Domain, and for the resident VPE, the IRS presents at most one candidate HPPI at a time. If the selected candidate HPPI changes, the IRS sends a replacement Forward command, which acts an implicit Recall if there is an outstanding Forward for the same Interrupt Domain.

A CPU interface might issue an Activate instead of a Release for the previous pending interrupt.

The protocol does not require that the new Forward is for an interrupt with higher priority than the outstanding Forward it replaces. A lower priority interrupt might replace a higher priority interrupt. For example, if an LPI is being re-targeted while it is pending, it must be retrieved from the old target CPU interface before being presented to the new target CPU interface. If there are other pending interrupts for the old target, one of these might now be presented to the CPU interface and this interrupt could be lower priority than the interrupt being re-targeted.

$I_{TZBHV}$ There is no limit on the number of Forward commands that can be outstanding.

$R_{KNCCF}$ For a given Interrupt Domain, there can never be multiple outstanding Forward commands for the same INTID.

$I_{CLNKP}$ If the IRS needs to re-issue a pending interrupt with new properties, for example a different priority value, it must first fully retrieve the previous pending interrupt.

$R_{GGHVV}$ When issuing a Forward command, the IRS sets `Returnable` based on the routing mode of the pending interrupt being forwarded:

- Targeted: `Returnable` is set to 0.
- 1ofN: It is IMPLEMENTATION DEFINED whether `Returnable` is set to 0 or 1.
  - Arm recommends that an implementation does not set `Returnable` to 1 unless there are other valid targets for the interrupt.

$R_{HWRFH}$ Forward commands are acknowledged by either an Activate command or Release command. For a given Interrupt Domain, or for the resident VPE, Forward commands are acknowledged in order.

A single Release command can acknowledge multiple Forward commands. The Release command acknowledges the INTID specified in the command, and all earlier unacknowledged Forward commands for the same Interrupt Domain or resident VPE.

$I_{HJMSR}$ For example, the CPU interface will acknowledge all the Forward commands for physical interrupts for the Non-secure Domain in the order it received them. But Forward commands for different Domains might be acknowledged out of order with respect to each other.

$R_{MYXCZ}$ For edge-triggered interrupts, receiving an ActivateAck command from the IRS in response to an Activate command guarantees that an edge generated by an interrupt source as a direct or indirect result of an instruction executed on the PE after observing the ActivateAck will not be merged into the acknowledged instance of the interrupt.

$R_{XSDST}$ The CPU interface issues Release commands in response to:

- A Recall command.
- A Forward that has been replaced by a subsequent Forward for the same Interrupt Domain.
- The most recent Forward for an Interrupt Domain had `Returnable` set to 1 and the CPU interface has for IMPLEMENTATION DEFINED reasons decided to return the interrupt.

$R_{MPTHB}$ For a given Interrupt Domain, or for the resident VPE, a Recall command applies to the most recent Forward command.

$R_{PKSKS}$ If a Recall command is received when there is no outstanding Forward command, the command is ignored.

$R_{GMQMJ}$ An IRS is able to accept an Activate or Release command for any outstanding Forward.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{TJKMB}$ The GICv5 Stream Protocol has no way for an IRS to signal that a Release or Activate has failed. Therefore, an IRS only forwards an interrupt to a CPU interface if it can accept either an Activate or Release for that interrupt.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.6 INTID configuration

$R_{DHHRZ}$ For any of the following the commands, when the IRS acknowledges the command, the IRS has observed the command and the effects of the command will be globally observable in finite time:

- Activate
- Deactivate
- SetEnabled
- SetHandling
- SetPending
- SetPriority
- SetTarget

$R_{DTMNP}$ The CPU interface issues an upstream Sync command to synchronize the effects of earlier commands.

The IRS responds with a Sync Ack once all the following are true:

- Any Activate, Deactivate, SetEnabled, SetHandling, SetPending, SetPriority, or SetTarget command received before the Sync has been acknowledged.
- The effects of any Activate, Deactivate, SetEnabled, SetHandling, SetPending, SetPriority, or SetTarget command received before the Sync are globally observable.

$R_{RXYSV}$ Between sending Sync and receiving Sync Ack, the CPU interface sends no commands other than to acknowledge downstream commands.

$I_{CPYQK}$ Although Activate commands acknowledge forward commands, the CPU interface cannot send an Activate between sending Sync and receiving Sync Ack. A Sync command is sent in response to execution of a `GSB SYS` barrier instruction, the barrier would require the CPU interface to have already sent an Activate from an earlier `GIC CDIA` and prevent execution of later `GIC CDIA` until the barrier was resolved.

$I_{PLQBY}$ The CPU interface might send an UpstreamControl command with `Identifier` set to Reset between sending a Sync and receiving the Sync Ack.

Arm expects resetting an online channel to be rare and be due to the PE resetting. If an online channel is reset, it is possible that some stale downstream commands are received from the IRS. The CPU interface ignores such commands.

$I_{MDVSS}$ The CPU interface can request the current configuration and state of an INTID by sending a RequestConfig command. The IRS returns the requested INTID's configuration when acknowledging the command with RequestConfig Ack.

$R_{DBJFG}$ Between sending RequestConfig and receiving RequestConfig Ack, the CPU interface sends no commands other than Sync commands or to acknowledge downstream commands.

The CPU interface does not send Activate commands between sending RequestConfig and receiving RequestConfig Ack. If a candidate HPPI has been acknowledged, the CPU interface sends the required Activate command before sending RequestConfig.

$I_{GLXSG}$ The GICv5 ordering rules mean that observing a prior change to an INTID's configuration via requesting its configuration requires that the effect of the change is observable to the requesting PE. Therefore, if the configuration being returned means that the INTID is not eligible to be the candidate HPPI, it must not be pending on the CPU interface when the RequestConfig Ack is sent.

$I_{GSGBS}$ The CPU interface might send an UpstreamControl command with `Identifier` set to Reset between sending a RequestConfig and receiving the RequestConfig Ack.

Arm expects resetting an online channel to be rare and be due to the PE resetting. If an online channel is reset, it is possible that some stale downstream commands are received from the IRS. The CPU interface ignores such commands.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A4.7 IRS and CPU interface capabilities

$I_{MNVFZ}$ The GICv5 Stream interface supports independent development of an IRS and a PE containing a CPU interface. This could lead to mismatches between the capabilities of the IRS and the connected PEs.

$R_{CQTBC}$ The CPU interface and IRS connected by an Interrupt Handling channel implement the same set of Interrupt Domains. Neither the CPU interface nor IRS generate any command with the `Domain` field set to an unimplemented Interrupt Domain.

$I_{MTKNY}$ Support for virtualization is reported in the IRS by IRS_IDR0.VIRT and in the PE by ID_AA64PFR0_EL1.EL2.

$I_{TGHLW}$ In GICv5, either both the IRS and CPU interface support virtualization or neither support virtualization.

$I_{PDCJG}$ GICv5 supports up to 5 bits of interrupts priority. The number of supported priority bits is reported by the IRS in IRS_IDR1.PRI_BITS and by the CPU interface in ICC_IDR0_EL1.PRI_BITS.

In GICv5 Stream, if the component receiving a command implements fewer than 5 bits of priority, the unimplemented least significant priority bits in the command are ignored and treated as being 0.

If the CPU interface and IRS implement different number of priority bits, Arm recommends that software restricts its usage to only those bits supported by both components.

$I_{RJJYX}$ GICv5 supports up to 24 bits of INTID.ID namespace for each interrupt type. The number of supported INTID.ID bits is reported by the IRS in IRS_IDR2.ID_bits and by the CPU interface in ICC_IDR0_EL1.ID_Bits. When fewer than 24 bits of identifier bits are supported the unimplemented bits are the most significant bits.

Arm recommends that all components in a GICv5 system implement the same number of INTID identifier bits. If different components support different number of INTID bits, Arm recommends that software restricts its usage to only those bits supported by all components.

$R_{YGCKH}$ If the IRS receives a command with INTID.ID beyond the implemented range, the command is acknowledged but otherwise ignored.

$R_{LQZNQ}$ If the CPU interface receives a Forward command with INTID.ID beyond the implemented range, the command is treated as an implicit Recall of any outstanding Forward for the specified Interrupt Domain or resident VPE, but is otherwise ignored.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter A5
# Interrupt Signaling channel

RETIRED

$I_{TJQNR}$ Interrupt Signaling channels allows interrupt sources to be connected to an IRS, and are intended as an alternative to using dedicated interrupt signals or MSIs.

$I_{HRNDT}$ The rules in this section only apply to the Interrupt Signaling channel.

$I_{GSNZT}$ How interrupts delivered by an Interrupt Signaling channel map to SPIs or IWB inputs is IMPLEMENTATION SPECIFIC.

$I_{QBKQL}$ Each Interrupt Source connected via an Interrupt Signaling channel can have one or more distinct interrupts.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A5.1 Command summary

$I_{WSFCN}$ The upstream commands that the Interrupt Source can send to the IRS are:

| ID | Command | Description |
|---|---|---|
| 0x0 | Reset | Resets the channel and deasserts all interrupts. |
| 0x1 | INT | Communicates the current state of an interrupt. |
| 0x2 | Resample Ack | Acknowledges a Resample command. |
| 0x3 | Quiesce | Requests the channel is taken offline. |
| 0x4 | Flush Ack | Acknowledges a Flush command. |
| 0x7 | IMPLEMENTATION DEFINED | Reserved for IMPLEMENTATION DEFINED functionality. |

All other IDs are reserved.

$I_{YMHFW}$ The downstream commands that an IRS can send to the Interrupt Source are:

| ID | Command | Description |
|---|---|---|
| 0x0 | Reset Ack | Acknowledges a Reset command. |
| 0x1 | - | |
| 0x2 | Resample | Requests current state of interrupt or interrupt levels. |
| 0x3 | Quiesce Ack | Acknowledges a Quiesce command. |
| 0x4 | Flush | Flushes commands from the IRS as part of a reset sequence. |
| 0x7 | IMPLEMENTATION DEFINED | Reserved for IMPLEMENTATION DEFINED functionality. |

All other IDs are reserved.

$R_{DNTTB}$ ID 0x7 in upstream and downstream is reserved for IMPLEMENTATION DEFINED functionality.

When supported, IMPLEMENTATION DEFINED commands do not impact the functionality of other commands. If not supported, IMPLEMENTATION DEFINED commands are ignored.

Arm strongly recommends that IMPLEMENTATION DEFINED commands are not used unless it can be guaranteed that both the sender and receiver agree on the usage.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A5.2 Outstanding commands

$I_{QBJBL}$ The following commands require acknowledgement:

| Command | Direction | Acknowledged by |
|---|---|---|
| Reset | Upstream | Reset Ack |
| Resample | Downstream | Resample Ack or Reset |
| Quiesce | Upstream | Quiesce Ack |
| Flush | Downstream | Flush Ack |

All other commands do not require acknowledgement.

$R_{FFYPH}$ Commands which require acknowledgement are acknowledged in finite time.

$I_{BBGBS}$ For commands that require acknowledgement, the architecture places limits on the number of outstanding commands that are permitted.

$R_{SWNBP}$ For each Interrupt Signaling channel, the IRS can have:

- Up to one Resample command outstanding.
- Up to one Flush command outstanding.

$R_{HNKHF}$ For each Interrupt Signaling channel, the CPU interface can have:

- Up to one Quiesce command outstanding.

$I_{QNLWZ}$ See A5.4 *Connection management* for more information about outstanding Reset commands.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A5.3 Signaling interrupts to the IRS

$R_{NVLNT}$ An Interrupt Source can support up to 256 interrupts, numbered 0 to 255.

$R_{GYFBQ}$ An Interrupt Source sends an INT command to indicate the state of an interrupt:

- `INT.ID` specifies the interrupt whose state is being communicated.
- `INT.Edge` and `INT.Level` indicate the current state of the interrupt.

| Edge | Level | Meaning | Example usage |
|---|---|---|---|
| 0 | 0 | The interrupt is de-asserted. | Used to indicate the interrupt has become de-asserted. |
| 0 | 1 | The interrupt is asserted. | Used to respond to Resample commands. |
| 1 | 0 | The interrupt was asserted and became de-asserted. | Used for edge-triggered interrupts to indicate a pulse. |
| 1 | 1 | The interrupt is asserted and remains asserted. | Used to indicate the interrupt has become asserted. |

$I_{CDSPZ}$ `INT.Edge` is used to identify positive edges, it is not used to indicate negative edges.

$I_{SRMND}$ An Interrupt Source is not required to know which interrupts have level or edge semantics. When the Interrupt Source knows that an interrupt has edge semantics, the {Edge,Level} value of {1,0} is more efficient than sending two INT commands to represent the interrupt becoming asserted then de-asserted. However, it remains permissible for the Interrupt Source to send two separate commands.

$R_{DBMYY}$ When an interrupt changes state, the Interrupt Source sends an INT command in finite time.

$R_{BMDPL}$ When the state of an interrupt changes multiple times before an INT command is sent, all of the following are true:

- If the interrupt is level-sensitive, the Interrupt Source is permitted to only present the final state of the interrupt.
- If the interrupt is edge-triggered, the Interrupt Source is permitted to only present a single edge.
- If the interrupt trigger-mode is unknown, the Interrupt Source is permitted to present a single edge and the final state.

$I_{MPJMH}$ Arm recommends that an interrupt source does not send INT commands unless the interrupt has changed state or it has received a Resample command.

$R_{TVKMG}$ It is IMPLEMENTATION DEFINED which values of `INT.ID` that an Interrupt Source may send.

$R_{TLDXR}$ The IRS can request the current state of an interrupt or of all interrupts by issuing a Resample command.

On receiving a Resample command:

- If `Resample.IDV` is 0:
  - The Interrupt Source sends an INT command for any interrupt source that is currently asserted, then sends a Resample Acknowledge command.
  - Any interrupt, for which the Interrupt Source does not send an INT command, between the Resample and Resample Acknowledge, is treated by the IRS as being de-asserted.
- If `Resample.IDV` is 1:
  - If the interrupt identified by `Resample.ID` is invalid, the Interrupt Source sends a Resample Acknowledge command.
  - If the interrupt identified by `Resample.ID` is valid and asserted, the Interrupt Source sends an INT command, then sends a Resample Acknowledge command.
  - If the interrupt identified by `Resample.ID` is valid and not asserted, the Interrupt Source sends a Resample Acknowledge command.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$I_{WCDNP}$ The protocol does not restrict which combinations of {Edge, Level} the Interrupt Source can use in INT commands between receiving a Resample and sending a Resample Acknowledge. Typically, Arm expects that the Interrupt Source will respond to the Resample with INT.{Edge, Level} set to {0,1} if the interrupt is asserted or to not send an INT command if the interrupt is deasserted. However, the Interrupt Source is permitted to send other combinations of {Edge, Level}, for example if the interrupt changes state during Resample.

$I_{XWNVG}$ Between receiving a Resample and sending a Resample Acknowledge, the protocol does not restrict which interrupts an Interrupt Source can send INT commands for. INT commands for interrupts not covered by the Resample could be sent before the Resample Acknowledge is sent.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A5.4 Connection management

$D_{DFJWP}$ When an Interrupt Signaling channel is described as *online*, upstream and downstream commands can be sent between the IRS and Interrupt Source. When an Interrupt Signaling channel is described as *offline*, the only commands that can be sent are those to bring the channel online.

$R_{XSJTH}$ The Interrupt Source sending a Reset command sets all interrupts to de-asserted and acknowledges any outstanding downstream commands.

$R_{VRGRT}$ To bring an Interrupt Signaling channel online, or reset an online channel:

- The Interrupt Source sends a Reset command.
- One or more instances of:
  - The IRS sends a Flush command.
  - The Interrupt Source acknowledges the Flush command.
- The IRS acknowledges the Reset command.

Once the Reset command is acknowledged, the Interrupt Signaling channel is online.

$R_{FWSZJ}$ When a Reset command is acknowledged:

- There are no outstanding upstream or downstream commands.
- All interrupts are treated as de-asserted.

$I_{NDWCH}$ Under normal operation, a Reset command is sent only to bring the connection between the Interrupt Source and IRS online. In which case there are no other outstanding commands.

However, the Interrupt Source may send a Reset command when the channel is already online. Sending the Reset command resets the connection, de-asserting all interrupts and cancelling outstanding commands.

When an online connection is being reset, it is possible that the Interrupt Source receives downstream commands between sending the Reset and receiving the acknowledgement. The Interrupt Source does not respond to these commands and the IRS does not expect to receive responses.

$R_{CSVKH}$ Between sending a Reset command and the command being acknowledged, the Interrupt Source sends no commands other than to acknowledge a Flush command.

When the IRS receives a Reset command, within finite time the IRS will respond with a Flush command. Once it has sent the Flush command, the IRS sends no other commands until is has received Flush Ack and sent the Reset Ack.

$I_{KZXLD}$ The IRS might issue some other commands between receiving the Reset command and sending the Flush command. The Interrupt Source ignores any downstream commands it receives between sending the Reset command and receiving a Flush command.

$R_{TMFQN}$ When the IRS sends a Flush command, the Sequence field is set to a value based on an IMPLEMENTATION DEFINED scheme.

When the Interrupt Source acknowledges a Flush, the Sequence field in the Flush Acknowledge command is set to the same value as provided in the Flush being acknowledged.

$R_{CPNXY}$ If the Interrupt Source receives multiple Flush commands, it is IMPLEMENTATION DEFINED whether it acknowledges each command or only the most recent. If the Interrupt Source acknowledges each command, it acknowledges them in the order they were received.

$R_{ZNKNG}$ The IRS's response to receiving a Flush acknowledge depends on the received Sequence number and whether there is an outstanding Flush.

The Flush acknowledge is ignored if either of the following is true:

- There is no outstanding Flush.
- The received Sequence number does not match that sent in the most recent Flush.

Otherwise, if none of the above are true, the IRS does one of:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- Send a Flush command.
- Send an acknowledge to the Reset command completeing the reset sequence.

$R_{GDBXW}$ To take an Interrupt Signaling channel offline:

- The Interrupt Source sends a Quiesce command.
  - The Interrupt Source sends no further commands, other than acknowledgements to downstream commands, until Quiesce Ack is received.
- The IRS completes any outstanding commands and then sends Quiesce Ack.
  - Once the IRS has received an upstream Quiesce command it does not send further Resample commands.

Once the IRS responds with Quiesce Acknowledge, the channel is offline.

$I_{WMGGK}$ It is possible that the Interrupt Source receives a downstream Resample command after the upstream Quiesce command is sent. The Interrupt Source is required to send Resample Ack before the IRS can respond with Quiesce Ack.

Taking an Interrupt Source channel offline implicitly deasserts all the interrupts. Therefore, if there is an outstanding Resample when Quiesce is issued, Arm recommends that the Interrupt Source sends Resample Ack without sending any INT commands, which indicates that all interrupts are deasserted. It is however permitted for the Interrupt Source to send INT commands and then Resample Ack.

$R_{YWLGB}$ If an Interrupt Signaling channel is offline, all of the following are true:

- The IRS sends no commands.
- The Interrupt Source sends no commands other than Reset to bring the channel online.
- All interrupts are treated as de-asserted.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter A6
# **Alphabetical list of commands**

RETIRED

This section contains the definition of each command for the GICv5 Stream Protocol.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A6.1 Interrupt Handling channel

This section contains the definition of each command for the Interrupt Handling channel for the GICv5 Stream Protocol.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

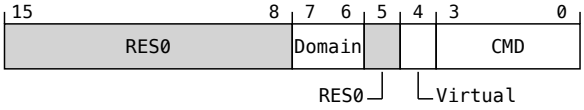
### A6.1.1 Activate, Activate command (CPUIF -> IRS)

The Activate characteristics are:

#### Attributes

Activate is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | Activate |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### Bit [5]

Reserved, RES0.

#### Domain, bits [7:6]

The Domain the operation affects.

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

#### Bits [15:8]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

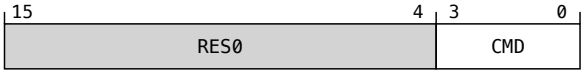
### A6.1.2 ActivateAck, Activate Acknowledge command (IRS -> CPUIF)

The ActivateAck characteristics are:

#### Attributes

ActivateAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0010 | Activate Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

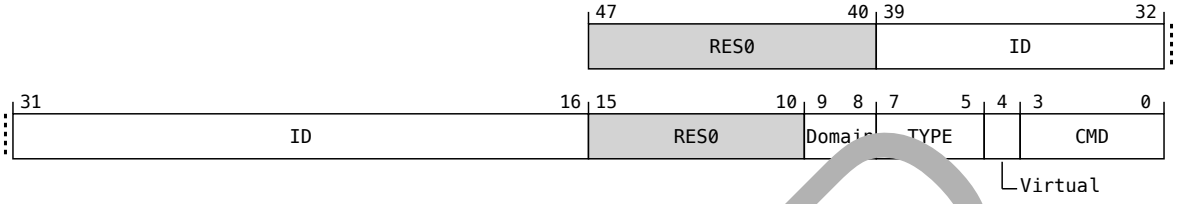
### A6.1.3 Deactivate, Deactivate interrupt command (CPUIF -> IRS)

The Deactivate characteristics are:

#### Attributes

Deactivate is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0110 | Deactivate |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### Bits [15:10]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

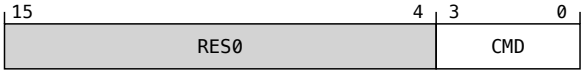
### A6.1.4 DeactivateAck, Deactivate interrupt Acknowledge command (IRS -> CPUIF)

The DeactivateAck characteristics are:

#### Attributes

DeactivateAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0110 | Deactivate Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

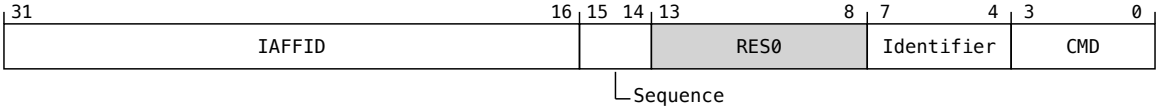
### A6.1.5 DownstreamControl, Downstream Control (IRS -> CPUIF)

The DownstreamControl characteristics are:

#### Attributes

DownstreamControl is a 4-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0101 | DownstreamControl |

#### Identifier, bits [7:4]

Specifies the type of Downstream Control operation.

| Identifier | Meaning |
|---|---|
| 0b0001 | Flush |
| 0b0010 | IAFFID |

#### Bits [13:8]

Reserved, RES0

#### Sequence, bits [15:14]

Flush sequence number

This field is RES0 when Identifier is not 0b0001 (Flush).

#### IAFFID, bits [31:16]

The PE interrupt affinity.

This field is RES0 when Identifier is not 0b0010 (IAFFID).

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

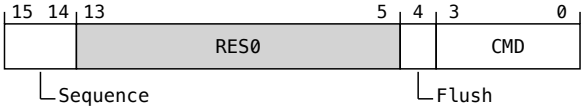
### A6.1.6 DownstreamControlAck, Downstream Control Acknowledge command (CPUIF -> IRS)

The DownstreamControlAck characteristics are:

#### Attributes

DownstreamControlAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0100 | DownstreamControl Ack |

#### Flush, bit [4]

Whether the operation being acknowledged is a Flush.

| Flush | Meaning |
|---|---|
| 0b0 | Not acknowledging a Flush. |
| 0b1 | Acknowledging a Flush. |

#### Bits [13:5]

Reserved, RES0

#### Sequence, bits [15:14]

Flush sequence number

This field is RES0 when Flush is not 1.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

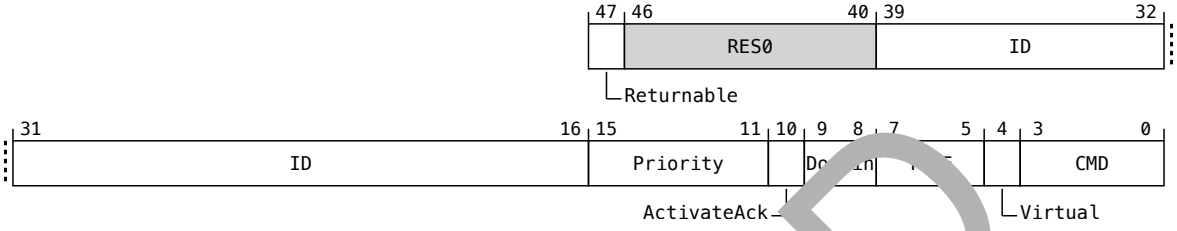
### A6.1.7 Forward, Forward command (IRS -> CPUIF)

The Forward characteristics are:

#### Attributes

Forward is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0001 | Forward |

#### Virtual, bit [4]

Whether the operation is on physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### ActivateAck, bit [10]

Whether the Forward also Acknowledges a previous Activate command.

| ActivateAck | Meaning |
|---|---|
| 0b0 | The Forward does not acknowledge any previous Activate. |
| 0b1 | The Forward acknowledges a previous Activate. |

When the Forward command acknowledges a previous Activate, it corresponds to an Activate for the same Domain and setting of Virtual.

### Priority, bits [15:11]

The priority of the interrupt.

Lower numeric values are higher priority, meaning that 0b00000 is the highest supported priority and 0b11111 is the lowest priority.

The CPU interface treats priority bits it does not implement as RES0.

The IRS sets priority bits it does not implement to 0b0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [46:40]

Reserved, RES0.

### Returnable, bit [47]

Indicates whether the CPU interface can choose to proactively return the interrupt.

| Returnable | Meaning |
|---|---|
| 0b0 | Interrupt is not returnable. |
| 0b1 | Interrupt is returnable. |

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

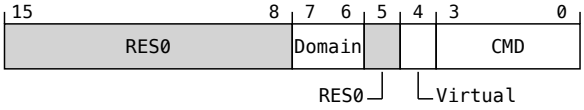
### A6.1.8 Recall, Recall command (IRS -> CPUIF)

The Recall characteristics are:

#### Attributes

Recall is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | Recall |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### Bit [5]

Reserved, RES0.

#### Domain, bits [7:6]

The Domain the operation affects.

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

#### Bits [15:8]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

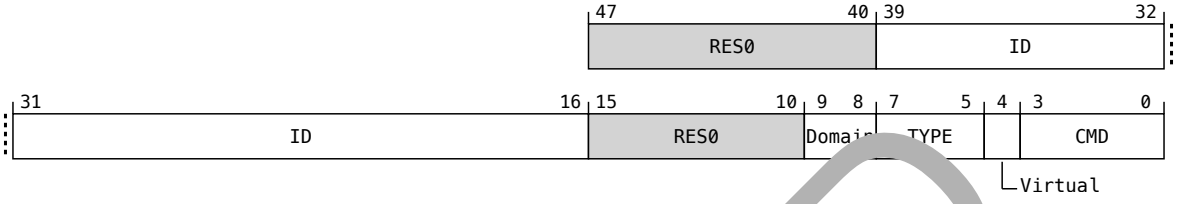
### A6.1.9 Release, Release command (CPUIF -> IRS)

The Release characteristics are:

#### Attributes

Release is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0011 | Release |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### Bits [15:10]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

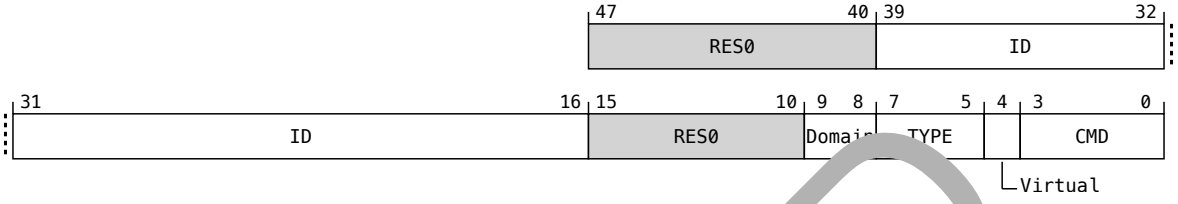
### A6.1.10 RequestConfig, Request Interrupt Configuration command (CPUIF -> IRS)

The RequestConfig characteristics are:

#### Attributes

RequestConfig is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1100 | RequestConfig |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### Bits [15:10]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

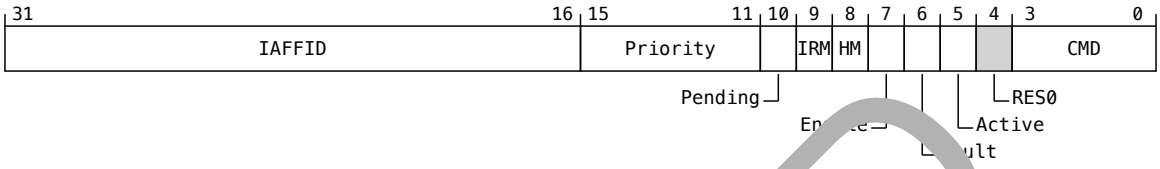
### A6.1.11 RequestConfigAck, Request Interrupt Configuration Acknowledge command (IRS -> CPUIF)

The RequestConfigAck characteristics are:

#### Attributes

RequestConfigAck is a 4-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1100 | RequestConfigAck |

#### Bit [4]

Reserved, RES0.

#### Active, bit [5]

Whether the interrupt is Active.

| Active | Meaning |
|---|---|
| 0b0 | Inactive |
| 0b1 | Active |

#### Fault, bit [6]

Whether the IRS was able to retrieve the requested data.

| Fault | Meaning |
|---|---|
| 0b0 | Success |
| 0b1 | Fault |

When this field is set to 1 it indicates the requested INTID was unreachable, the returned configuration is UNKNOWN.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### Enable, bit [7]

Whether the interrupt is individually enabled.

| Enable | Meaning |
|---|---|
| 0b0 | Disabled |
| 0b1 | Enabled |

### HM, bit [8]

Whether the interrupt is Level or Edge.

| HM | Meani |
|---|---|
| 0 | Edge |
| 0b1 | L el |

### IRM, bit [9]

Interrupt routing mode

| IRM | Meaning |
|---|---|
| 0b0 | Interrupt Routing mode is Targeted. |
| 0b1 | Interrupt Routing mode is 1ofN. |

### Pending, bit 0]

Whether the interrup pending

| Pending | Meaning |
|---|---|
| 0b0 | Not pending. |
| 0b1 | Pending. |

### Priority, bits [15:11]

The priority of the interrupt.

### IAFFID, bits [31:16]

The interrupt Affinity value.

When IRM is 1, this field is IMPLEMENTATION SPECIFIC.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

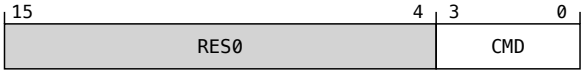
### A6.1.12 SetAck, Set interrupt configuration acknowledge command (IRS -> CPUIF)

The SetAck characteristics are:

#### Attributes

SetAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1000 | Set Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

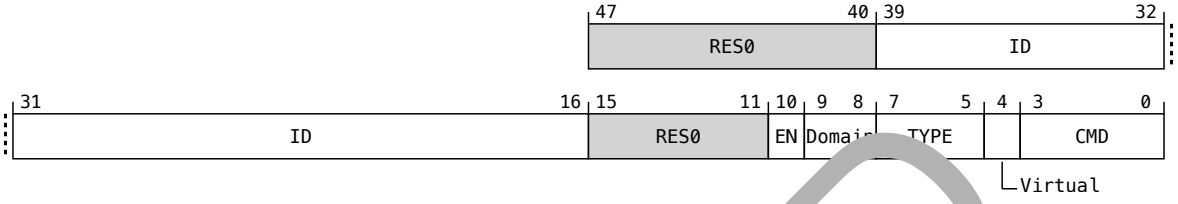
### A6.1.13 SetEnabled, Set interrupt Enabled command (CPUIF -> IRS)

The SetEnabled characteristics are:

#### Attributes

SetEnabled is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1000 | SetEnabled |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### EN, bit [10]

Whether the interrupt is Enabled.

| EN | Meani |
|---|---|
| 0b0 | Disabled |
| 0b1 | E d |

### Bits [15:11]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

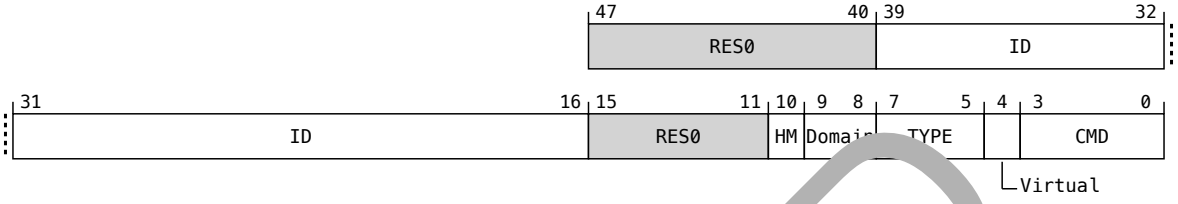
### A6.1.14 SetHandling, Set Interrupt Handling Mode command (CPUIF -> IRS)

The SetHandling characteristics are:

#### Attributes

SetHandling is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1011 | SetHandling |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### HM, bit [10]

Interrupt Handling mode.

| HM | Meaning |
|---|---|
| 0b0 | Edge |
| 0b1 | [illegible] |

### Bits [15:11]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

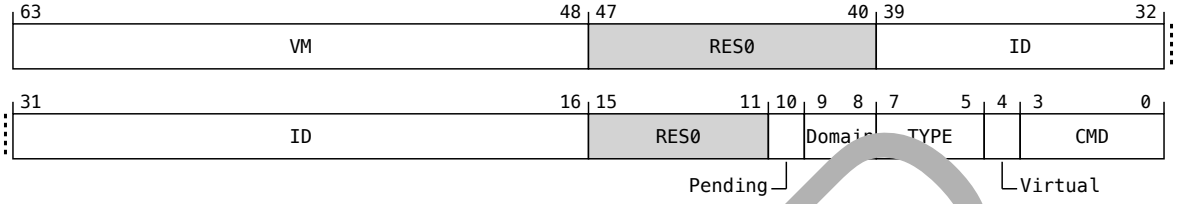
### A6.1.15 SetPending, Set interrupt Pending command (CPUIF -> IRS)

The SetPending characteristics are:

#### Attributes

SetPending is a 8-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1110 | SetPending |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### Pending, bit [10]

Whether the interrupt is Pending.

| Pending | Meaning |
|---|---|
| 0b0 | Generate CLEAR event. |
| 0b1 | Generate SET event. |

### Bits [15:11]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

### VM, bits [63:48]

The Virtual Machine identifier. This field is RES0 when Virtual is 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

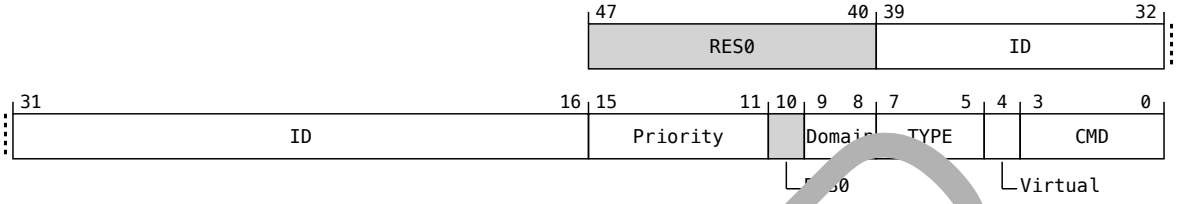
### A6.1.16 SetPriority, Set Interrupt Priority command (CPUIF -> IRS)

The SetPriority characteristics are:

#### Attributes

SetPriority is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1010 | SetPriority |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### Bit [10]

Reserved, RES0.

### Priority, bits [15:11]

The priority of the interrupt.

The CPU interface sets priority bits it does not implement to 0b

The IRS treats priority bits it does not implement as 0b0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

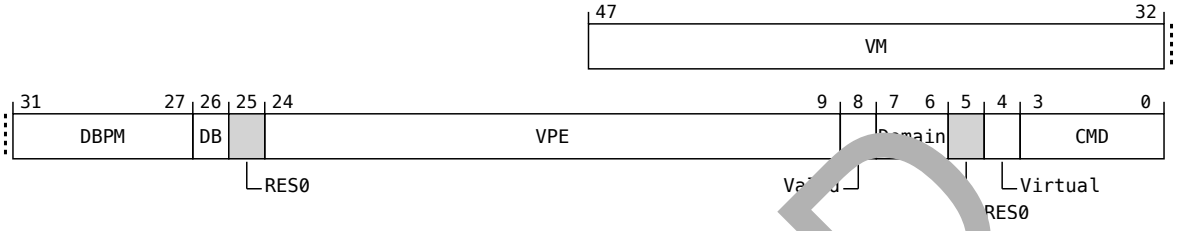
### A6.1.17 SetResident, Set Resident command (CPUIF -> IRS)

The SetResident characteristics are:

#### Attributes

SetResident is a 6-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1111 | SetResident |

#### Virtual, bit [4]

Whether the operation is physical or virtual.

| Virtual | Meaning |
|---|---|
| 0b1 | Virtual interrupt. |

#### Bit [5]

Reserved, RES0.

#### Domain, bits [7:6]

The Interrupt Domain of the VPE being made resident or non-resident.

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b11 | Realm |

#### Valid, bit [8]

Whether a VPE is resident or not.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Valid | Meaning |
|---|---|
| 0b0 | The currently resident VPE is being cleared. |
| 0b1 | The VPE indicated by the Domain and VPE fields is being made resident. |

### VPE, bits [24:9]

The VPE identifier. This field is RES0 when Valid is 0.

### Bit [25]

Reserved, RES0.

### DB, bit [26]

Doorbell Request.

| DB | Meaning |
|---|---|
| 0b0 | No doorbell requested |
| 0b1 | Doorbell requested |

This field is RES0 when Valid is 1.

### DBPM, bits [31:27]

Doorbell Priority Mask.

This field is RES0 when DB is 0.

### VM, bits [47:32]

The Virtual Machine Identifier. This field is RES0 when Valid is 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

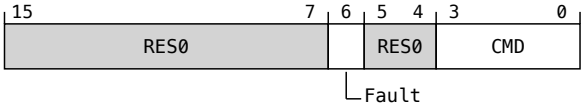
### A6.1.18 SetResidentAck, Set Resident acknowledge command (IRS -> CPUIF)

The SetResidentAck characteristics are:

#### Attributes

SetResidentAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1[illegible] | SetResident Ack |

#### Bits [5:4]

Reserved, RES0.

#### Fault, bit [6]

Whether the requested VPE was made resident.

| Fault | Meaning |
|---|---|
| 0b0 | Requested VPE was made resident. |
| 0b1 | Requested VPE was not made resident. |

When acknowledging a SetResident with Valid set to 0, this field is RES0.

#### Bits [7]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

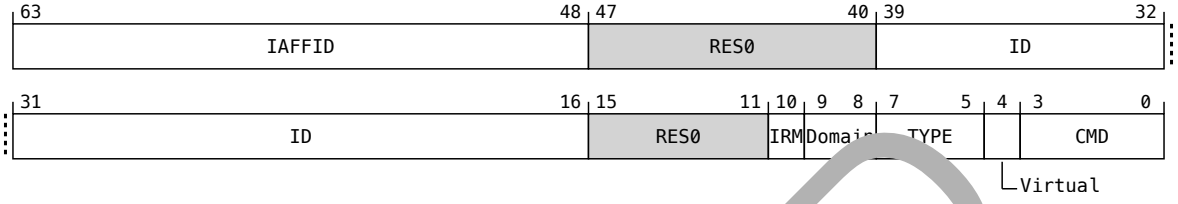
### A6.1.19 SetTarget, Set Interrupt Target command (CPUIF -> IRS)

The SetTarget characteristics are:

#### Attributes

SetTarget is a 8-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1001 | SetTarget |

#### Virtual, bit [4]

Whether the operation is on a physical or virtual interrupt.

| Virtual | Meaning |
|---|---|
| 0b0 | Physical interrupt. |
| 0b1 | Virtual interrupt. |

#### TYPE, bits [7:5]

The type of the interrupt.

| TYPE | Meaning |
|---|---|
| 0b010 | LPI |
| 0b011 | SPI |

Values not defined above are reserved.

#### Domain, bits [9:8]

The Domain the operation affects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Domain | Meaning |
|---|---|
| 0b00 | Secure |
| 0b01 | Non-secure |
| 0b10 | EL3 |
| 0b11 | Realm |

### IRM, bit [10]

Interrupt routing mode

| IRM | Meaning |
|---|---|
| 0b0 | The interrupt Routing mode is Targeted. |
| 0b1 | The interrupt Routing mode is 1ofN. |

### Bits [15:11]

Reserved, RES0.

### ID, bits [39:16]

The ID of the interrupt.

### Bits [47:40]

Reserved, RES0.

### IAFFID, bits [63:48]

Target PE.

If IRM is 0, this field indicates the PE interrupt affinity of the target PE.

If IRM is 1, this field provides an IMPLEMENTATION DEFINED hint to 1ofN selection algorithm. A value of 0 means that no hint is provided.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

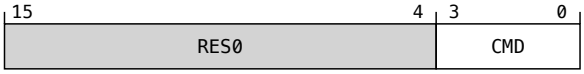
### A6.1.20 Sync, synchronizes previously sent configuration changes (CPUIF -> IRS)

The Sync characteristics are:

#### Attributes

Sync is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1101 | Sync |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

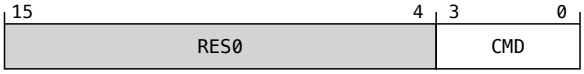
### A6.1.21 SyncAck, synchronizes previously sent configuration changes (IRS -> CPUIF)

The SyncAck characteristics are:

#### Attributes

SyncAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b1101 | Sync Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

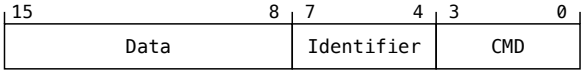
### A6.1.22 UpstreamControl, Upstream Control (CPUIF -> IRS)

The UpstreamControl characteristics are:

#### Attributes

UpstreamControl is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | UpstreamControl |

#### Identifier, bits [7:4]

Specifies the type of Upstream Control operation.

| Identifier | Meaning |
|---|---|
| 0b0000 | Reset |
| 0b0001 | Quiesce |
| 0b0010 | Set 1ofN hints, Data specifies which hints are being set. |
| 0b0011 | Clear 1ofN hints, Data specifies which hints are being cleared. |

#### Data, bits [15:8]

Data associated with the Upstream Control operation.

When Identifier is 0b0010 or 0b00011, Data indicates which 1ofN hints are being set or cleared.

| Data | Meaning |
|---|---|
| 0bxxxxxxx1 | Selecting this PE is likely to cause exit from a low power state. Applies to all Domains. |

Values not defined above are reserved.

For all other values of Identifier, this field is RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

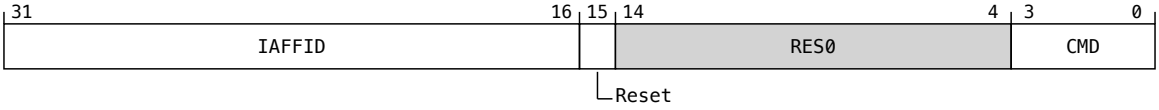
### A6.1.23 UpstreamControlAck, Upstream Control Acknowledge command (IRS -> CPUIF)

The UpstreamControlAck characteristics are:

#### Attributes

UpstreamControlAck is a 4-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | UpstreamControl Ack |

#### Bits [14:4]

Reserved, RES0.

#### Reset, bit [15]

Whether a reset request is being acknowledged.

| Reset | Meaning |
|---|---|
| 0b0 | Not acknowledging a reset request. |
| 0b1 | Acknowledging a reset request. |

#### IAFFID, bits [31:16]

The PE interrupt affinity.

This field is RES0 when acknowledging an UpstreamControl with an Identifier that is not 0b0000 (Reset).

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

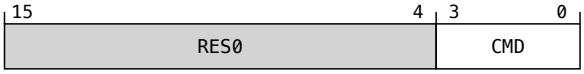
### A6.1.24 WakeRequest, Wake request (IRS -> CPUIF)

The WakeRequest characteristics are:

#### Attributes

WakeRequest is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0101 | WakeRequest |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A6.2 Interrupt Signaling channel

This section contains the definition of each command for the Interrupt Signaling channel for the GICv5 Stream Protocol.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

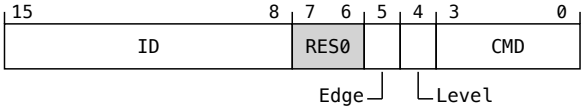
### A6.2.1 INT, Interrupt command (Interrupt Source -> IRS)

The INT characteristics are:

#### Attributes

INT is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | INT |

#### Level, bit [4]

| Level | Meaning |
|---|---|
| 0b0 | De-asserted. |
| 0b1 | Asserted. |

#### Edge, bit [5]

| Edge | Meaning |
|---|---|
| 0b0 | Falling-edge, or no edge. |
| 0b1 | Positive-edge. |

#### Bits [7:6]

Reserved, RES0.

#### ID, bits [15:8]

The ID of the interrupt.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

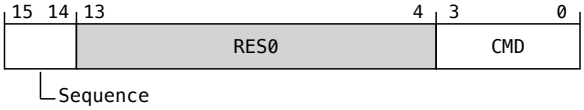
### A6.2.2 Flush, Flush command (IRS -> Interrupt Source)

The Flush characteristics are:

#### Attributes

Flush is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0001 | Flush |

#### Bits [13:4]

Reserved, RES0.

#### Sequence, bits [15:14]

Flush sequence number

This field is RES0 when Identifier is not 0b0001 (Flush).

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

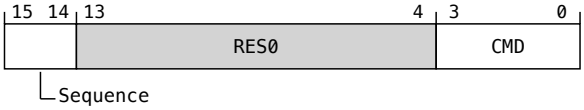
### A6.2.3 FlushAck, Flush acknowledge command (Interrupt Source -> IRS)

The FlushAck characteristics are:

#### Attributes

FlushAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| [illegible] | Flush Ack |

#### Bits [13:4]

Reserved, RES0.

#### Sequence, bits [15:14]

Flush sequence number

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

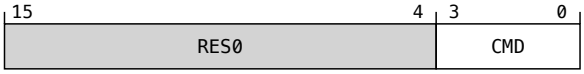
### A6.2.4 Quiesce, Quiesce command (Interrupt Source -> IRS)

The Quiesce characteristics are:

#### Attributes

Quiesce is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0011 | Quiesce |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

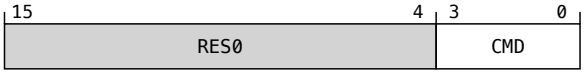
### A6.2.5 QuiesceAck, Quiesce acknowledge command (IRS -> Interrupt Source)

The QuiesceAck characteristics are:

#### Attributes

QuiesceAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0011 | Quiesce Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

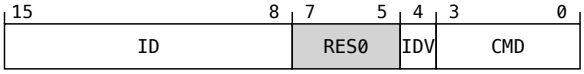
### A6.2.6 Resample, Resample request command (IRS -> Interrupt Source)

The Resample characteristics are:

#### Attributes

Resample is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0010 | Resample |

#### IDV, bit [4]

| IDV | Meaning |
|---|---|
| 0b0 | Resample request for all asserted interrupts. |
| 0b1 | Resample request for the interrupt identified by ID field. |

#### Bits [7:5]

Reserved, RES0.

#### ID, bits [15:8]

The ID of the interrupt. This field is RES0 when IDV is 0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

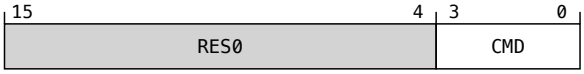
### A6.2.7 ResampleAck, Resample request acknowledge command (Interrupt Source -> IRS)

The ResampleAck characteristics are:

#### Attributes

ResampleAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0010 | Resample Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

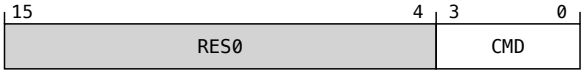
### A6.2.8 Reset, Reset command (Interrupt Source -> IRS)

The Reset characteristics are:

#### Attributes

Reset is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | Reset |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

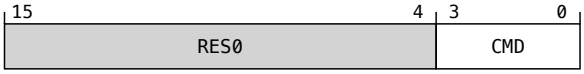
### A6.2.9 ResetAck, Reset acknowledge command (IRS -> Interrupt Source)

The ResetAck characteristics are:

#### Attributes

ResetAck is a 2-byte structure.

#### Field descriptions




#### CMD, bits [3:0]

Indicates the command type.

| CMD | Meaning |
|---|---|
| 0b0000 | Reset Ack |

#### Bits [15:4]

Reserved, RES0.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Chapter A7
# **Example sequences**

RETIRED

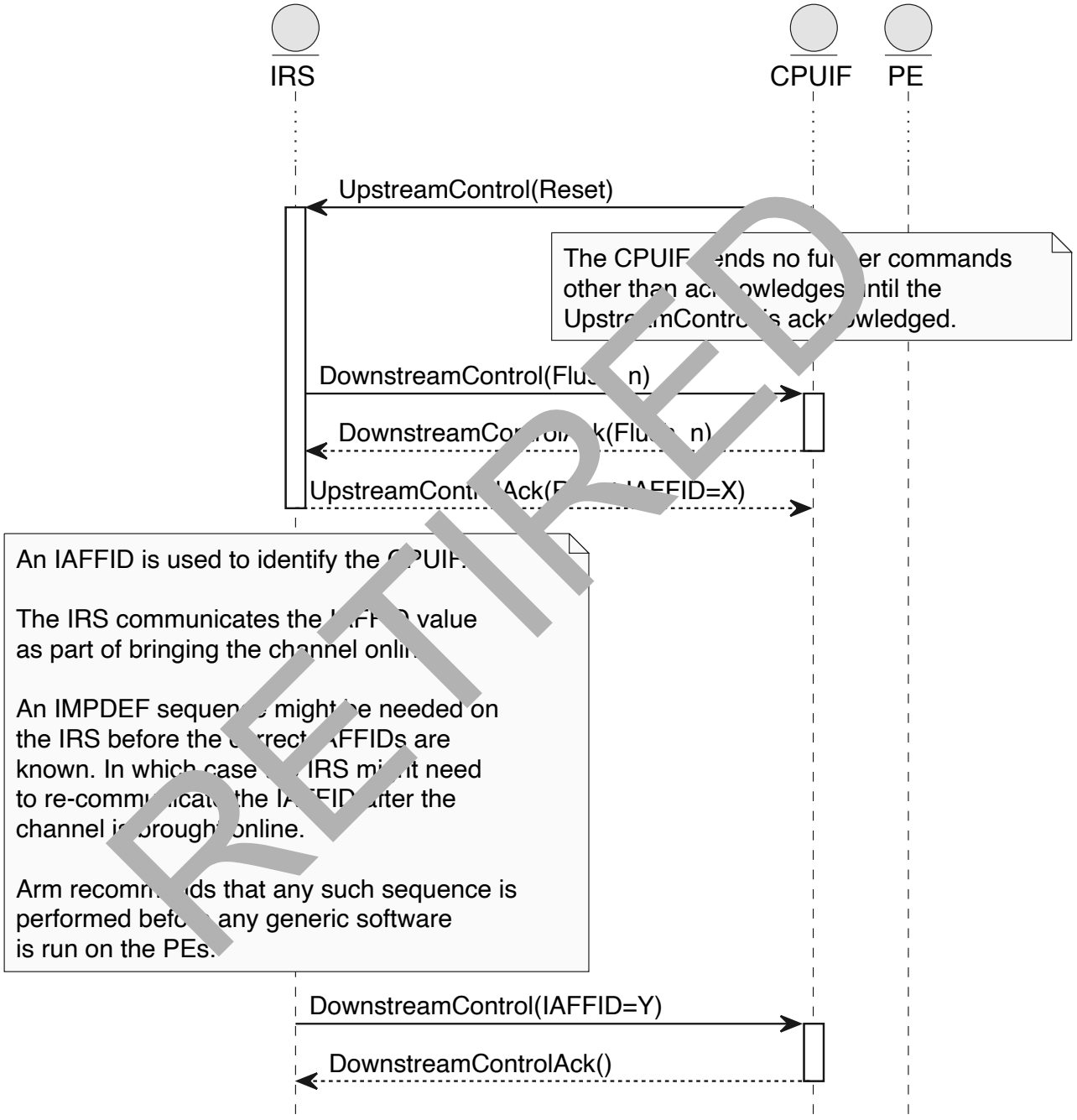
This section gives examples of typical sequences.

The flows given in this section are for illustration purposes only.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.1 Bringing the Interrupt Handling channel online and taking it offline

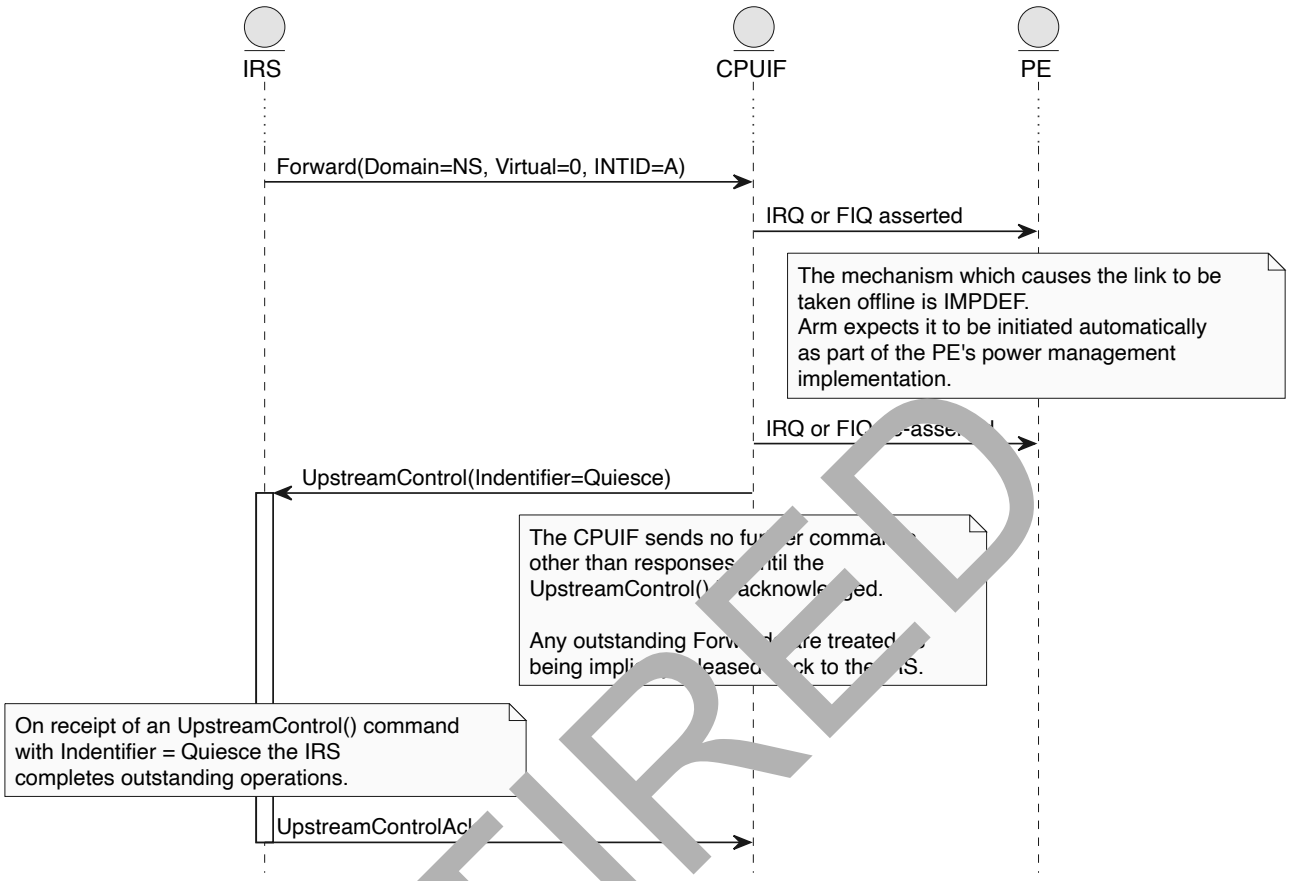
If the Interrupt Handling channel between the CPU interface and IRS is offline, the first command that is sent is an UpstreamControl command. A two-phase reset process is used, as illustrated below:




The Interrupt Handling channel is taken offline by the CPU interface sending an UpstreamControl command with Identifier set to Quiesce. On receipt of the UpstreamControl command the IRS completes all outstanding commands, before sending a UpstreamControlAck. The IRS treats all outstanding Forward commands as being implicitly returned by the link that is taken offline. Once the UpstreamControlAck is received by the PE the Interrupt Handling channel is offline.

RETIRED

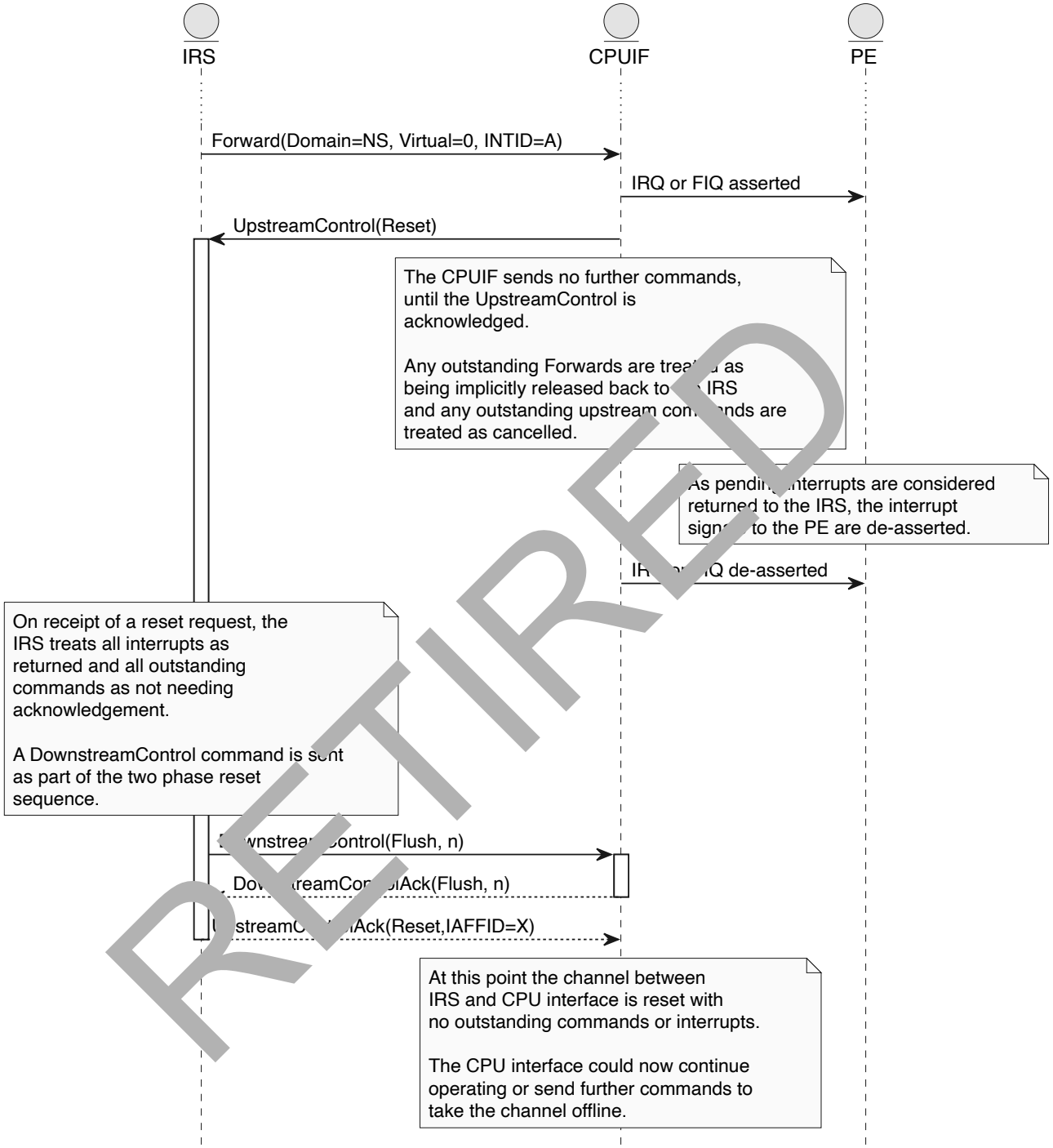

*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*




There may be cases where the Interrupt Handling channel is classed as online but needs to be reset. This can be achieved by the CPUIF interface sending an UpstreamControl command with Identifier set to Reset. Sending the UpstreamControl command with `Identifier` set to `Reset` returns any pending interrupts to the IRS and cancels outstanding commands. This process is illustrated below:

RETIRED

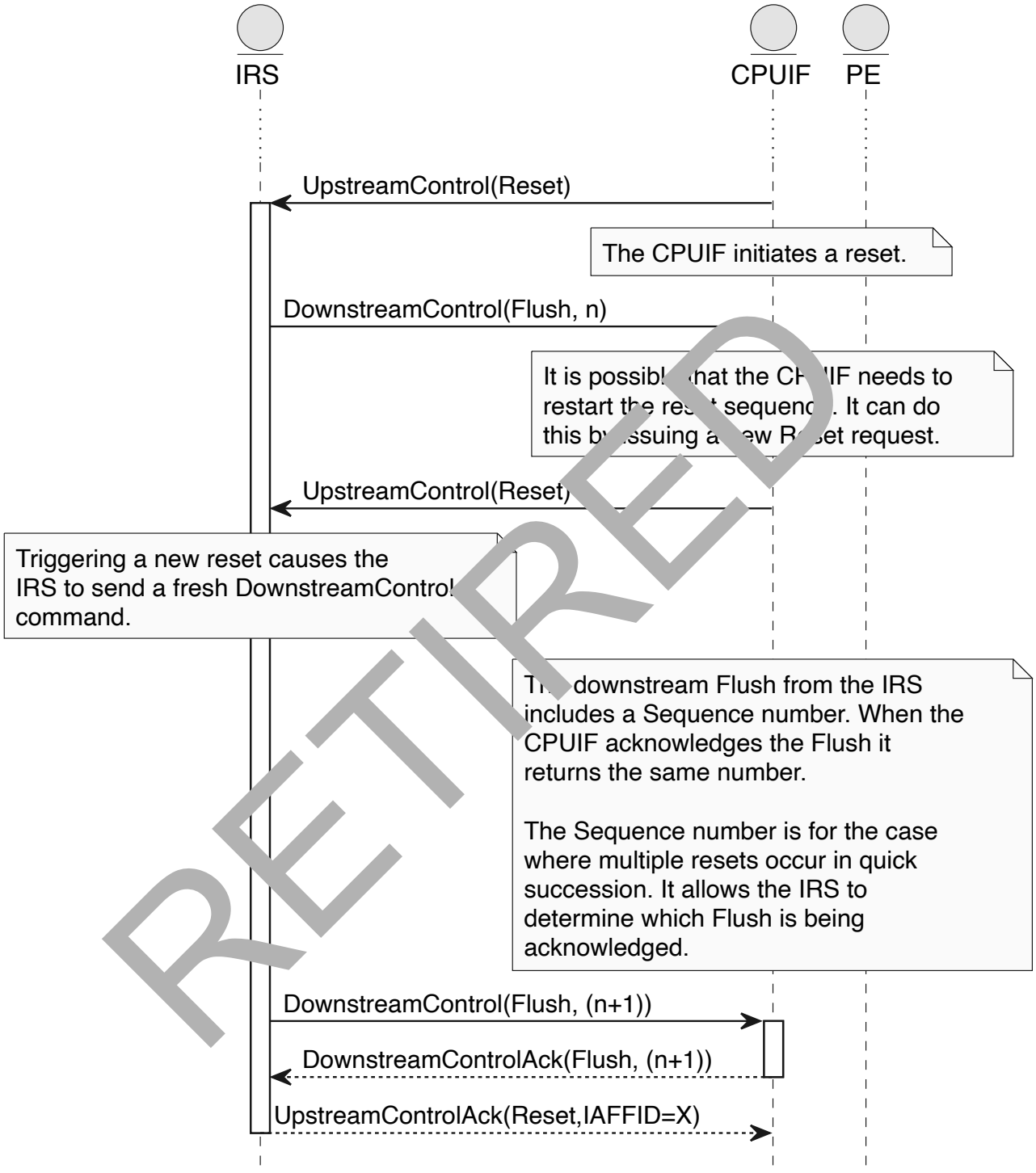

*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*




RETIRED

The CPU interface might need to restart the reset process part way through. To do this, the CPU interface sends a new Reset request.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

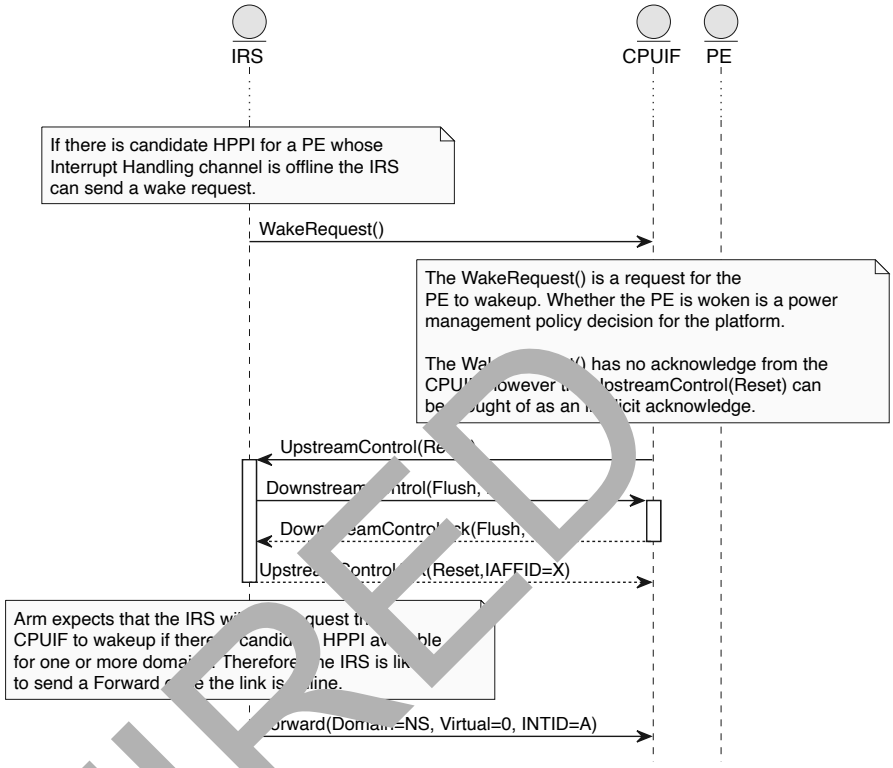



RETIRED

Arm recommends that resetting an online connection is only used for recovery and not for normal operation.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

If the Interrupt Handling channel is offline, the IRS can request that it is brought online by sending a wake request:

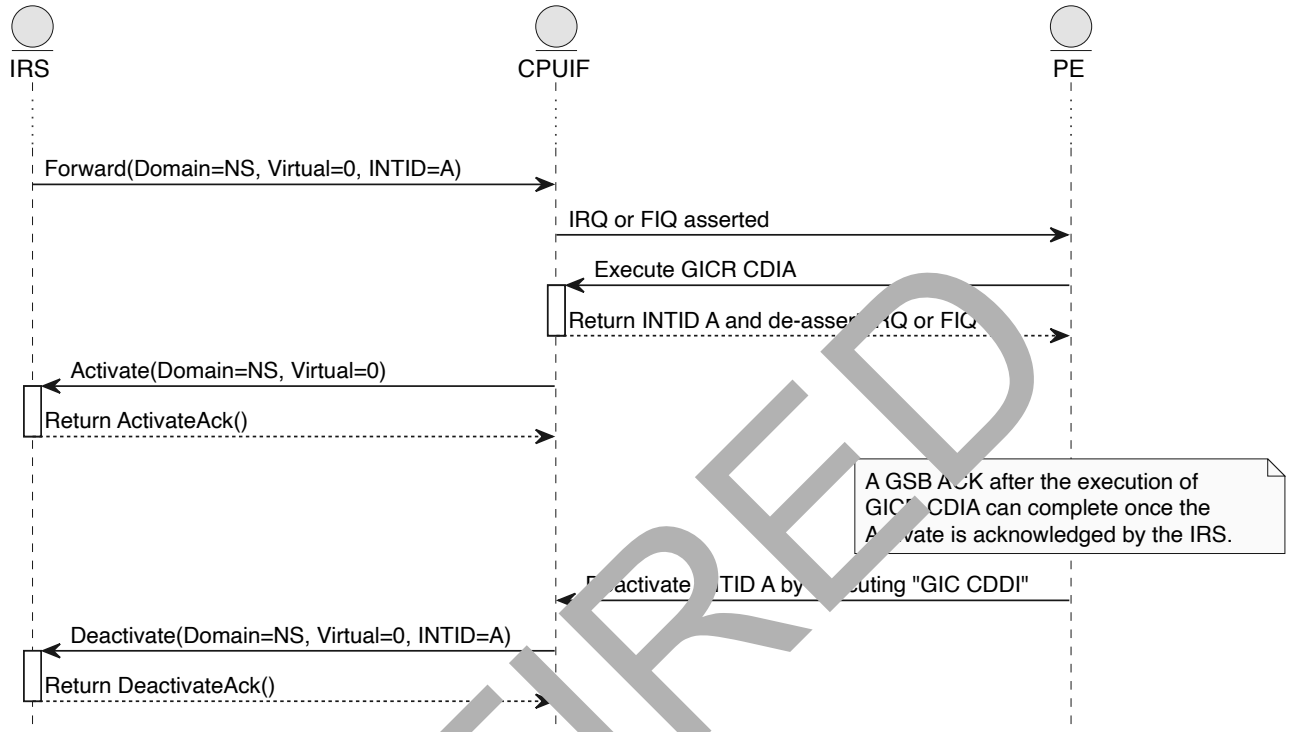



RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.2 Simple interrupt life-cycle

The diagram below shows a physical interrupt being presented to a PE by an IRS:




The effect of acknowled ing an interr is not guaranteed to have been observed by the IRS until after a `GSB ACK` or `GSB SYS` instructio has completed. W GICv5 Stream, a PE can determine the activation has been observed by waiting for an A tivateAck() before completing the barrier.

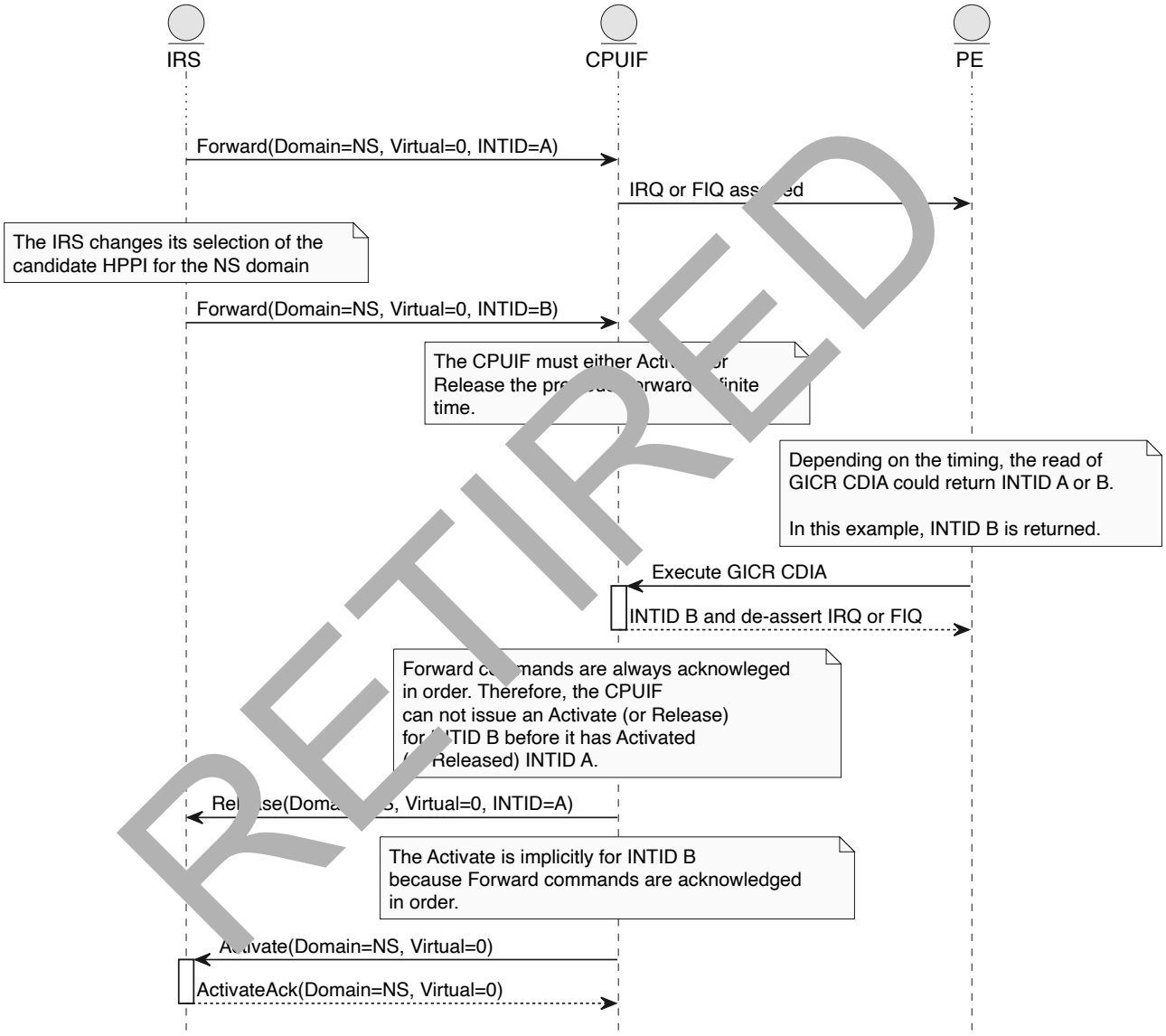
With edge-trigger interrupts, until the acknowledging of an instance of an interrupt is visible to the IRS, further edges from the interr source m ht continued to be merged.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.3 Replacing the candidate HPPI for an Interrupt Domain, or resident VPE

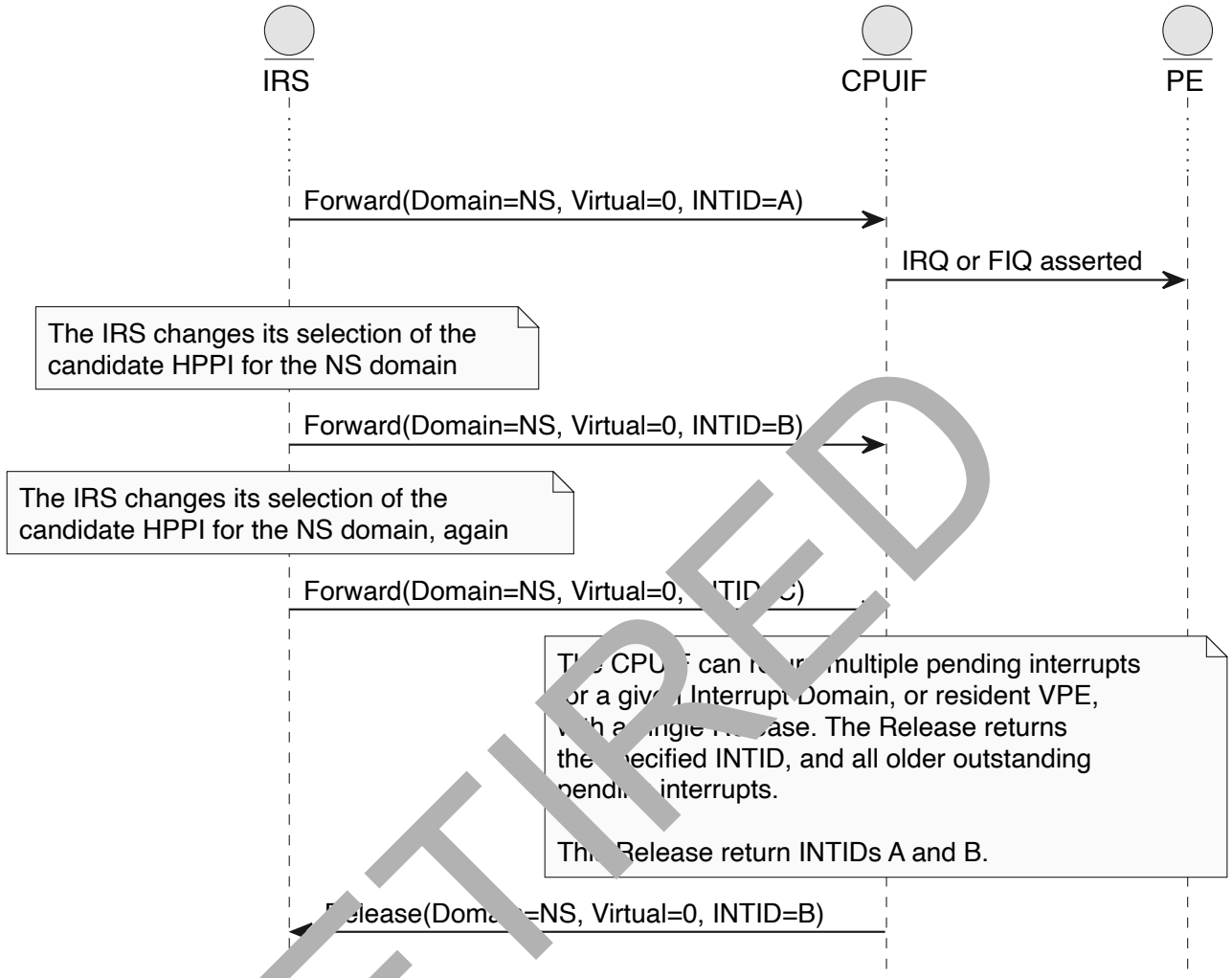
The IRS forwards the candidate HPPI for a given Interrupt Domain, or resident VPE, to the CPU interface. The chosen candidate HPPI might change, for example due to another interrupt becoming pending or a change to interrupt configuration. In this case, the IRS can replace the candidate HPPI for the Interrupt Domain, or resident VPE, by sending another Forward. The CPU interface is required to either release or activate the replaced interrupt in finite time.




RETIRED

It is possible that the IRS again changes the candidate HPPI before the CPU interface has responded in finite time. The IRS can send further Forward commands, each replacing the previous. The CPU interface can return multiple pending interrupts for a given Interrupt Domain, or VPE, with a single Release command. The Release command releases the specified INTID, and all earlier outstanding Forward commands for that Interrupt Domain, or VPE.


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

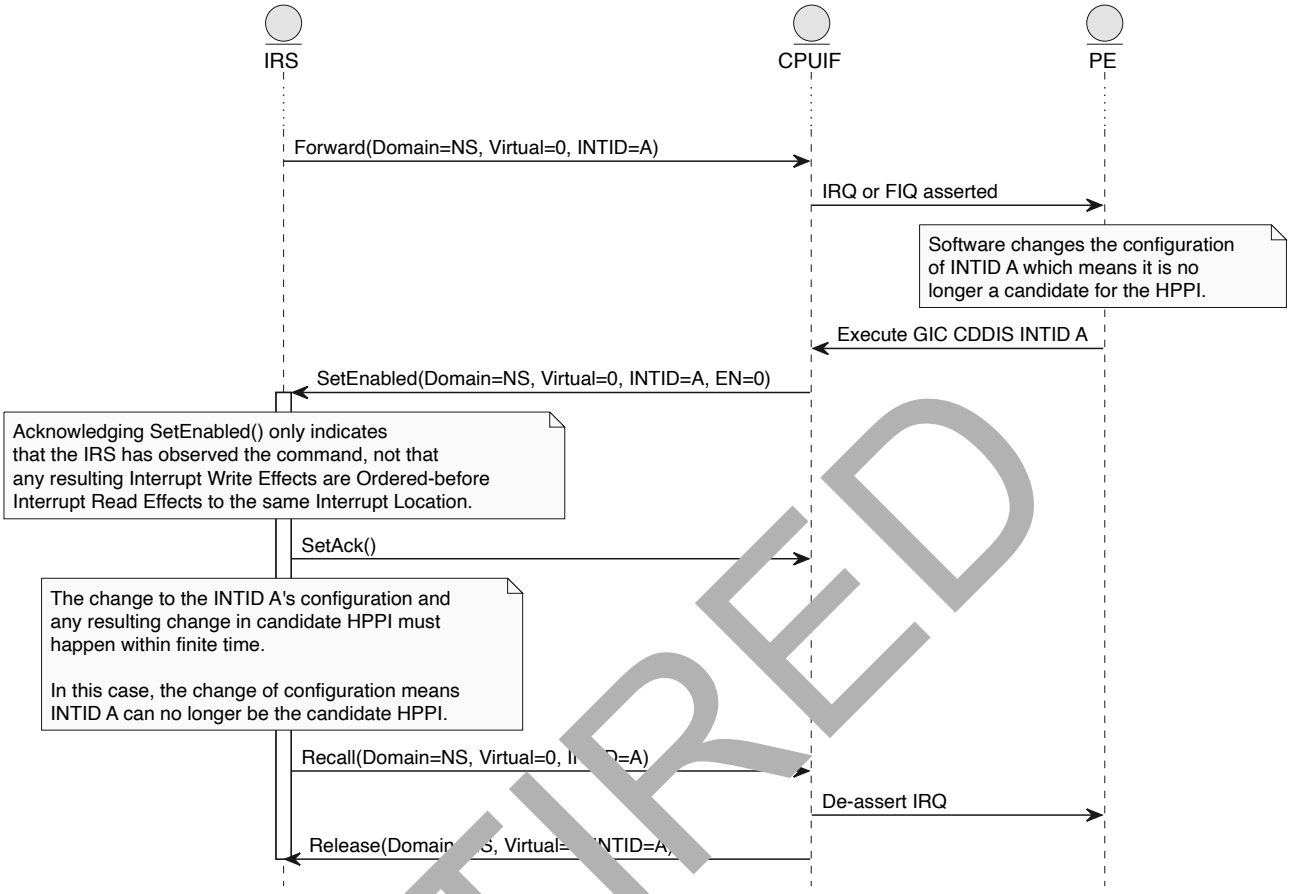



For a giv rupt D ain, o e resident VPE, Forward commands are always acknowledged in the order they are re ved. In e above ple, the CPU interface cannot issue an Activate for INTID C before INTIDs A and B a released.

In som es the candidate HPPI might change due to GIC instructions executed on the PE that the interrupt has been forw ed to. In these cases the CPU interface communicates the change in configuration to the IRS and the IRS is respon le for recalling the interrupt if required. The CPU interface does not proactively issue a Release.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*




In the above example, the IRS acknowledges the SetEnabled command before the Interrupt Write effect of clearing the INTID Enable is visible. Acknowledging a SetEnabled, SetPending, SetPriority or SetTarget indicates that the IRS has observed the command and that the effects of the command will become visible in finite time. To guarantee that the effects of previous operations are complete, software must issue a `GSB SYS` barrier which results in a SyncDone command being sent to the IRS.

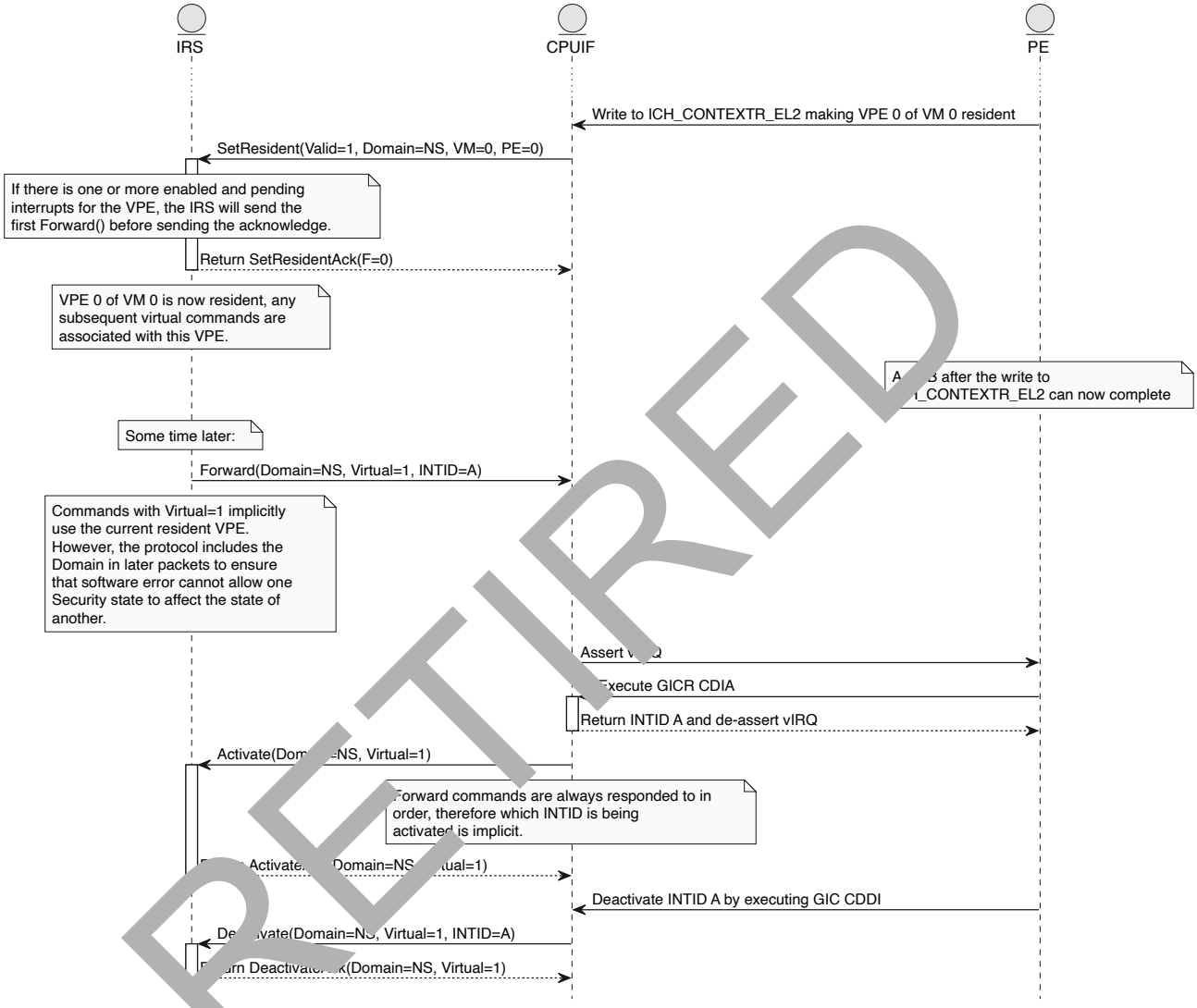
Another execution where the IRS sends the SetAck after the Interrupt Write effects are globally visible is also possible and permitted by the architecture.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.4 Making a VPE resident

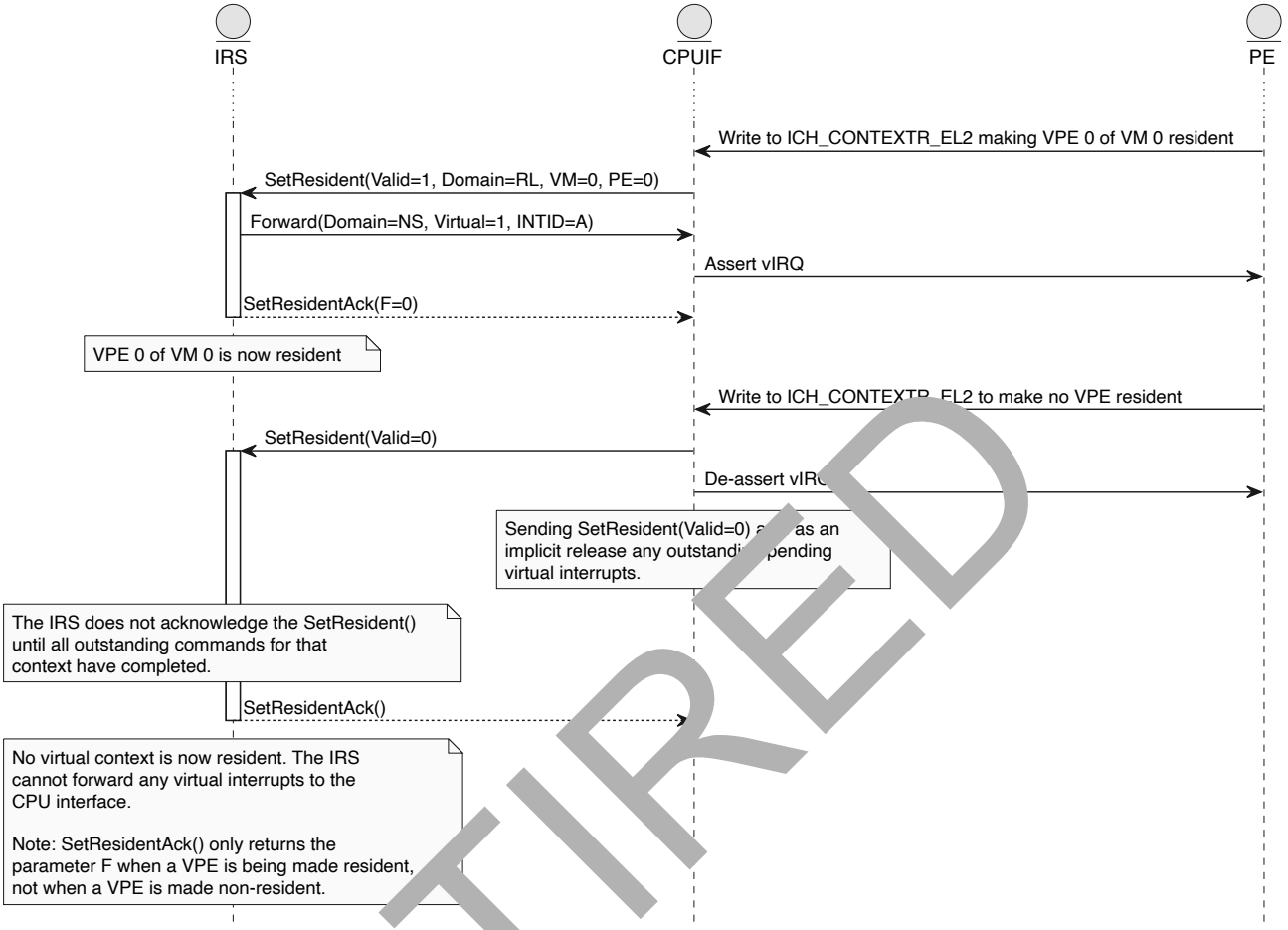
To set the resident VPE, the CPU interface sends a SetResident() command. The IRS responds with a SetResidentAck() to show that it has completed the operation. This is illustrated here:




RETIRED

Software can also configure the GIC so that there is no resident VPE, as illustrated here:

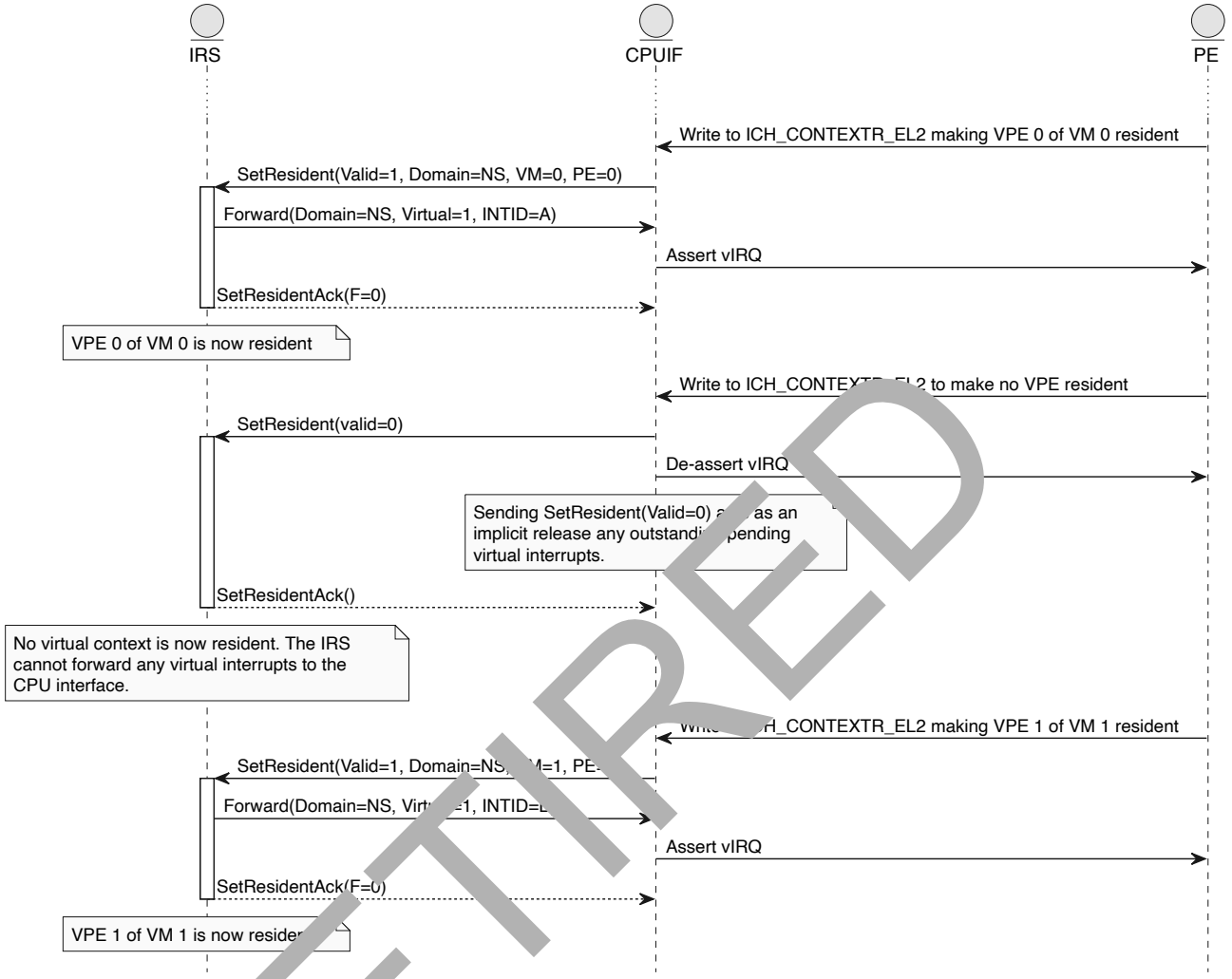

*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*



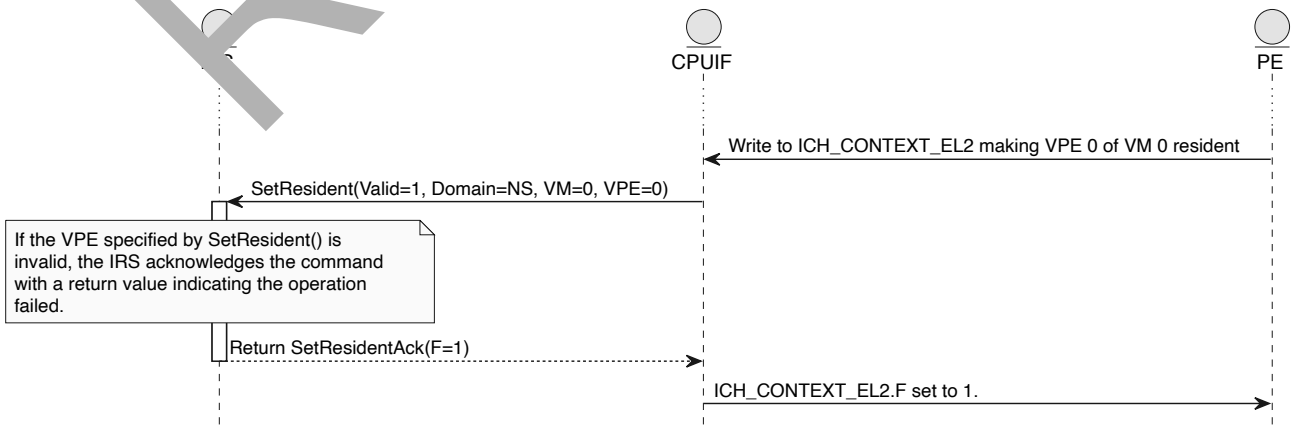

To change the resident VPE, the current resident VPE must first be made non-resident:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*




If the CPU interface attempts to make an invalid VPE resident, the IRS acknowledges the command indicating that operation failed.

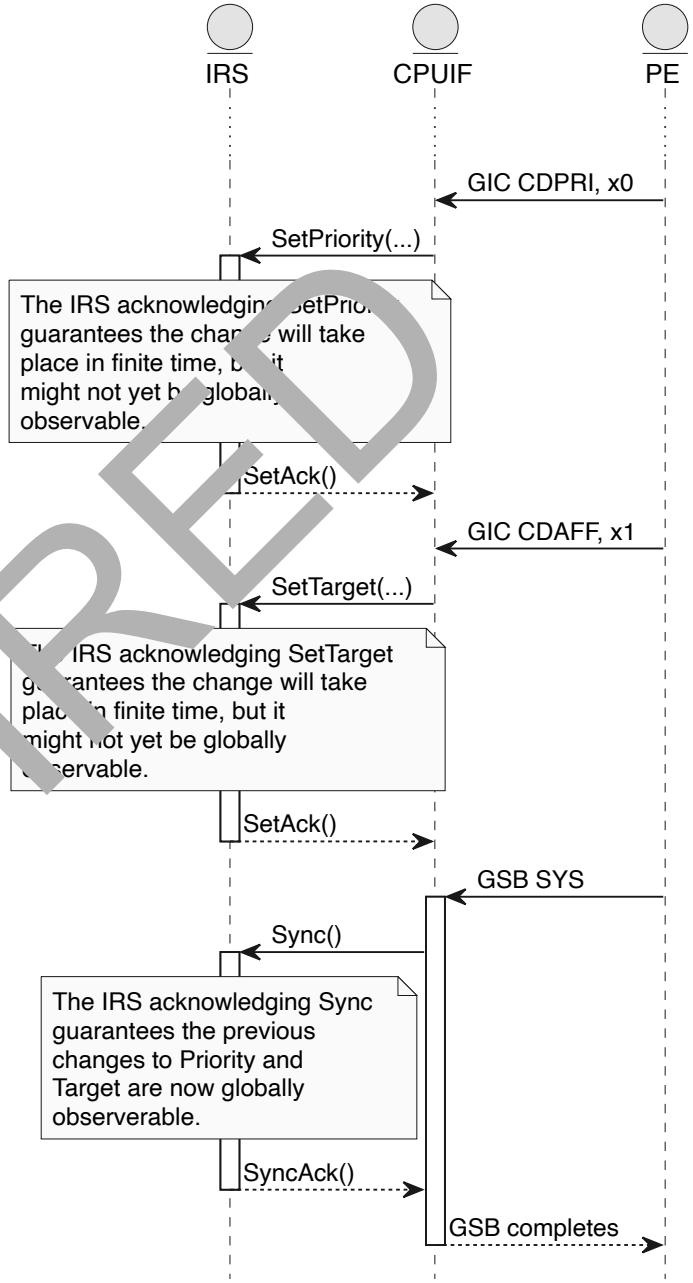



RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.5 Interrupt configuration

The flow for updating the configuration of an LPI or SPI is illustrated below:

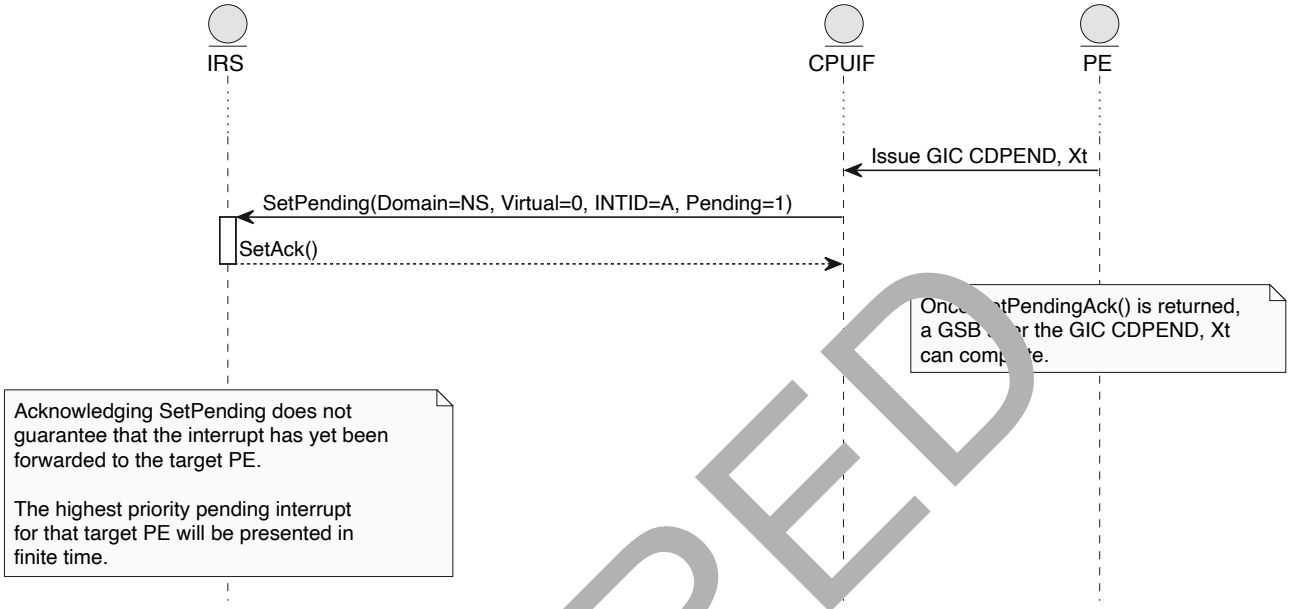



RETIRED

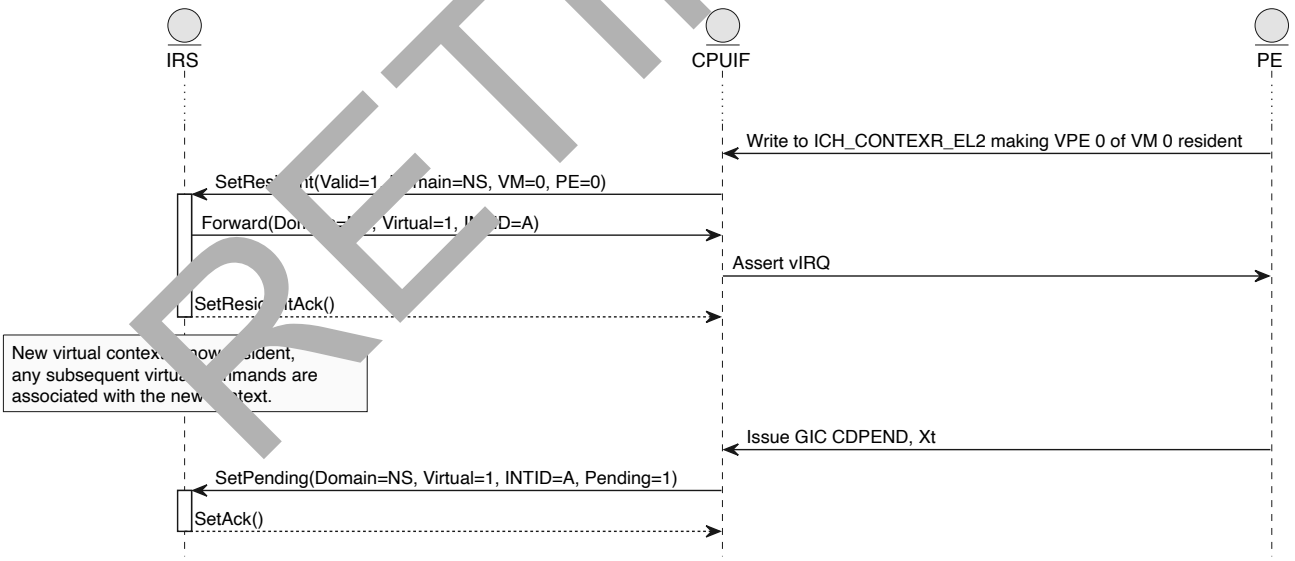

*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.6 Sending IPIs

The flow for generating a physical IPI is illustrated here:




For virtual IPIs, the VPE is taken from the previous SetResident() command, as illustrated here:

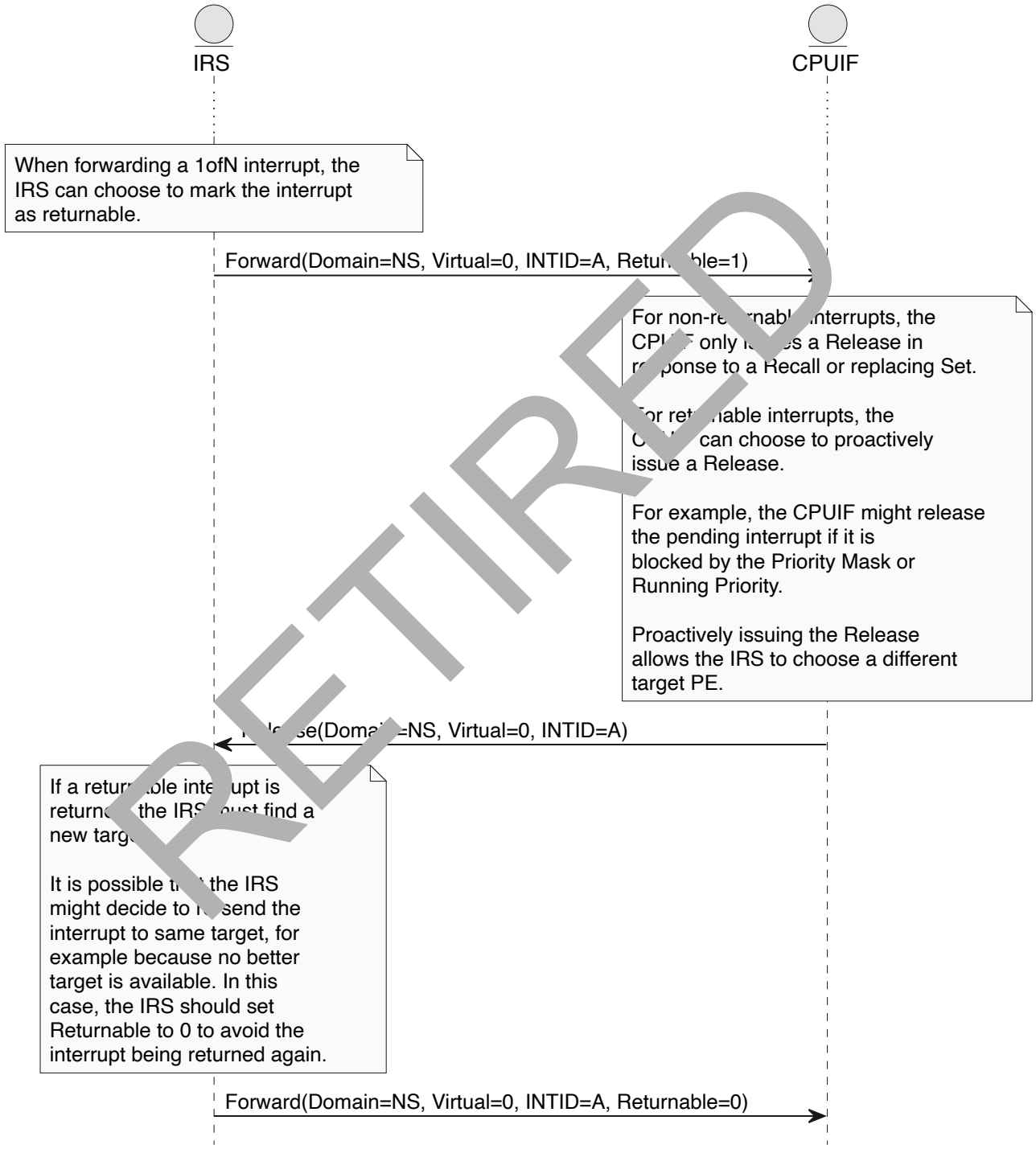



RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## A7.7 1 of N interrupts

The following flow demonstrates how the `Returnable` argument to a Forward command can be used for 1ofN interrupts:




RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

RETIRED

# Part B
## Litmus tests

# Chapter B1
# Interrupt ordering litmus tests

RETIRED

This section gives examples of the use of GIC instructions with and without the use of the GIC specific barrier instructions introduced by this document.

The aim of these tests is to help programmers, hardware design engineers, and validation engineers understand the need for the different kinds of barriers and inherent ordering guarantees.

See also:

- *B.12 Interrupt ordering model and synchronization requirements*


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## B1.1 Interrupt litmus test assumptions

The litmus tests make the following assumptions related to GIC instructions, configuration of the GICv5 CPU interface, and interrupt state and configuration:

- We assume all interrupts are initialized with the following values:

  ```
  [INTID(x)]=(pending:0, enabled:1, priority:1, target:P0, routing_mode:targeted,
  ↪handling_mode:edge)
  ```

- We assume no asynchronous events take place, including no interrupts become pending without explicit actions taken by one of the processes in the litmus test.
- We assume that ICC_PCR_EL1 and ICC_HAPR_EL1 are both set at the idle priority.
- We assume all GICv5 CPU interface and IRS controls are set to enable delivery of interrupts to PEs included in the litmus test.
- We assume that SCTLR_ELx.NMI is 1.
- We assume interrupt exceptions are masked using `PSTATE` including interrupts with Superpriority.
- We assume all INTIDs referenced from the litmus tests are reachable from the PEs.
- We assume all PEs are executing in the same Interrupt Domain.
- Interrupts are delivered in finite time.
- Some litmus tests rely on a peripheral with the following properties:
  - The peripheral generates interrupt event to INTID(A) as a result of writing 1 to an MMIO register on the device.
  - A mapping to the interrupt generating MMIO register is annotated with `//PERIP` in the test.
  - The register initially has the value 0 when read back.
  - The register preserves the value written to it.
  - The peripheral generates a single edge event when there is a write to its MMIO register.
  - The interrupt is made Pending in finite time after the write to the peripheral's MMIO register.
- The `WFI(instr, <cond>)` construct executes the instruction in a loop enough times that either `<cond>` is true, or an implementation-defined finite time periods have completed.
  - For example, `WFI(GICR WAIT(GICR Xt, CDIA; Xt=(valid:1))` will eventually acknowledge an interrupt if all of the following are true:
    - The interrupt is Pending or becomes Pending in finite time.
    - The interrupt is Inactive, targeted to the PE, and of sufficient Priority to be acknowledged.

See the documentation of the hertools7 toolsuite (https://diy.inria.fr/) for more information about the syntax and use of litmus tests.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.2 Atomicity of interrupt updates by GIC system instructions

## B1.2.1 Notes

A GIC system instruction generates a Successful Read-Modify-Write pair of INTID Effects to update an interrupt. The purpose of this section is to show that the Successful Read-Modify-Write pair is required to be atomic.

## B1.2.2 Litmus test

```
Arch64 gic.pri+gic.en
{
 [INTID(A)]=(priority:0,enabled:0);
 0:X1=(intid:A, priority:1);
 1:X1=(intid:A);
}
 P0            | P1           ;
 GIC CDPRI, X1 | GIC CDEN, X1 ;

exists(INTID(A)=(priority:0,enabled:1) \/ INTID(A)=(priority:1,enabled:0))

kinds: gic.pre+gic.en Forbid
```

### B1.2.2.1 Explanation

On P0, the `GIC CDPRI,X1` system instruction updates the priority of the interrupt `INTID(A)`:

- Generates an Interrupt Read Effect $E_1$ to `INTID(A)`.
- Modifies the `priority` field of the value read.
- Generates an Interrupt Write Effect $E_2$ to `INTID(A)`.

$E_1$ and $E_2$ are a Successful Read-Modify-Write pair of Interrupt Effects and have to be atomic.

On P1, the `GIC CDEN, X1` system instruction enables the interrupt `INTID(A)`:

- Generates an Interrupt Read Effect $E_3$ to `INTID(A)`.
- Modifies the `enabled` field of the value read.
- Generates an Interrupt Write Effect $E_4$ to `INTID(A)`.

$E_3$ and $E_4$ are a Successful Read-Modify-Write pair of Interrupt Effects and have to be atomic.

Therefore, an execution where $E_1$ is Coherence-before $E_4$, $E_4$ is Coherence-before $E_2$ and as a result `INTID(A)` has the final value `(priority:1, enabled:0)` is architecturally forbidden. Also, an execution where $E_3$ is Coherence-before $E_2$ and $E_2$ is Coherence-before $E_4$ and as a result `INTID(A)` has the final value `(priority:0, enabled:1)` is architecturally forbidden.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.3 Multiple updates of the same Interrupt Location

## B1.3.1 Notes

The purpose of this test is to show that multiple upadtes to the same Interrupt Location by GIC system instructions with commands other than `DI` are observed in program order.

## B1.3.2 Litmus test with two configuration updates

```
AArch64 coWW-gic
{
 [INTID(A)]=(priority:0);
 0:X1=(intid:A, priority:1);
 0:X2=(intid:A, priority:2);
}
 P0                    ;
 GIC CDPRI, X1         ;
 GIC CDPRI, X2         ;

exists(INTID(A)=(priority:0) \/ INTID(A)=(priority:1))

kinds: coWW-gic Forbid
```

### B1.3.2.1 Explanation

On P0:

- The first `GIC CDPRI,X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ that updates its priority.
- The second `GIC CDPRI,X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that updates its priority.

$E_1$ and $E_2$ are a Successful Read-Modify-Write pair of Interrupt Effects and are required to be atomic.

Therefore, an execution where $E_1$ is Coherence-before $E_4$, $E_4$ is Coherence-before $E_2$ and as a result `INTID(A)` has the final value `(priority:1, enabled:0)` is architecturally forbidden. Also, an execution where $E_3$ is Coherence-before $E_1$ and $E_2$ is Coherence-before $E_4$ and as a result `INTID(A)` has the final value `(priority:0,enabled:1)` is architecturally forbidden.

## B1.3.3 Litmus test with an interrupt disable and an interrupt deactivate

```
AArch64 coWR-gic+dis-di+rcfg
{
 [INTID(A)]=(enabled:1, active:1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0             | P1                    ;
 GIC CDDIS,X1   | GIC CDRCFG, X1        ;
 GIC CDDI,X1    | ISB                   ;
                | MRS X3, ICC_ICSR_EL1 ;
exists(1:X3=(active:0,enabled:1))

kinds: coWR-gic+dis-di+rcfg Allowed
```

### B1.3.3.1 Explanation

On P0:

- `GIC CDDIS,X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$.
- `GIC CDDI,X1` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

On P1:

- `GIC CDRCFG,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition, $E_5$ Reads-from $E_4$ and $E_4$ is Coherence-before $E_1$. This is architecturally allowed.

### B1.3.4 Litmus test with an interrupt deactivate and an interrupt disable

```
AArch64 coWR-gic+dis-di+rcfg
{
 [INTID(A)]=(enabled:1, active:1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0            | P1                  ;
 GIC CDDI,X1   | GIC CDRCFG, X1      ;
 GIC CDDIS,X1  | ISB                 ;
               | MRS X3, ICC_ICSR_EL1 ;
exists(1:X3=(active:1,enabled:0))

kinds: coWR-gic+di-dis+rcfg Allowed
```

#### B1.3.4.1 Explanation

On P0:

- `GIC CDDI,X1` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_2$.
- `GIC CDDIS,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$.

On P1:

- `GIC CDRCFG,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition, $E_5$ Reads-from $E_4$ and $E_4$ is Coherence-before $E_1$. This is architecturally allowed.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.4 Reading back interrupt writes on a single PE

## B1.4.1 Notes

The purpose of these tests is to show that writing the configuration of an interrupt or acknowledging an interrupt and reading back the configuration on the same PE guarantees that the written values are observed without the use of explicit synchronization.

## B1.4.2 Litmus test with a configuration update

```
AArch64 coWR-gic+en-rcfg
{
 [INTID(A)]=(enabled:0, affinity:P0);
 0:X1=(intid:A);
}
 P0                   ;
 GIC CDEN, X1         ;
 GIC CDRCFG, X1       ;
 ISB                  ;
 MRS X3, ICC_ICSR_EL1 ;

exists(0:X3=(affinity:P0,enabled:0))

kinds: coWR-gic+en-rcfg Forbid
```

### B1.4.2.1 Explanation

On P0:

- `GIC CDEN, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_2$.

In the execution that satisfies the postcondition, $E_2$ Coherence-before $E_1$. This violates the architectural requirements of the Coherence-before relation and as a result, this execution is architecturally forbidden.

## B1.4.3 Litmus test with deactivate

```
AArch64 coWR-gic+di-rcfg
{
 [INTID(A)]=(active:1);
 0:X1=(intid:A);
}
 P0                   ;
 GIC CDDI, X1         ;
 GIC CDRCFG, X1       ;
 ISB                  ;
 MRS X2, ICC_ICSR_EL1 ;

exists(0:X2=(active:1))

kinds: coWR-gic+di-rcfg Forbid
```

### B1.4.3.1 Explanation

On P0:

- `GIC CDDI, X1` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GIC CDRCFG, X1` generates an Interrupt Read Effect to `INTID(A)` $E_2$.

In the execution that satisfies the postcondition, $E_2$ Coherence-before $E_1$. This violates the architectural requirements of the Coherence-before relation and as a result, this execution is architecturally forbidden.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### B1.4.4 Litmus test with acknowledgement

```
AArch64 coWR-gic+ia-rcfg
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P0);
 0:X1=(intid:A);
}
 P0                        ;
 GICR    X2, CDIA          ;
 GIC     CDRCFG, X1        ;
 ISB                       ;
 MRS     X3, ICC_ICSR_EL1  ;

exists(0:X2=(valid:1) /\ 0:X3=(pending:1))

kinds: coWR-gic+ia-rcfg Forbid
```

#### B1.4.4.1 Explanation

On P0:

- `GIC X2, CDIA` generates an Implicit Interrupt Write Eff[illegible] to `INTID(A)` $E_1$.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Re[illegible] Effect t[illegible]`INTID(A)` [illegible].

In the execution that satisfies the postcondition, $E_2$ is Co[illegible]ce-befor[illegible] $E_1$. This violates the architectural requirements of the Coherence-beforerelation and[illegible]s a[illegible]ult, th[illegible]execu[illegible]n is architecturally forbidden.

### B1.4.5 Litmus test with two configuration [illegible]dates

```
AArch64 coWWR-gic+aff-en-rcfg
{
 [INTID(A)]=(enabled:0, affini[illegible]:P0);
 0:X0=(intid:A, affinity:P1[illegible]
 0:X1=(intid:A);
}
 P0
 GIC CDAFF, X0          ;
 GIC CDEN, X1           ;
 GIC CDRCFG, X[illegible]           ;
 ISB
 MRS X3, ICC_ICSR_[illegible]

exists [illegible]:X3=(a[illegible]finity:[illegible]e[illegible]bled:0) \/
       0:X3=(a[illegible]finity:P1,[illegible]nabled:0) \/
       0:X3=[illegible]:P0,enabled:1))

kinds: c[illegible]WR-gic+aff-en-rcfg Forbid
```

#### B1.4.5.1 E[illegible]planation

On P0:

- `GIC CDAFF, X0` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GIC CDEN, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_2$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_3$.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_4$.

In the executions that satisfy the postcondition, any of the following applies:

- $E_4$ is Coherence-before $E_1$. This violates the architectural requirements of the Coherence-before relation and as a result, this execution is architecturally forbidden.
- $E_2$ is Coherence-before $E_1$ and E4~ Reads-from $E_3$. This violates the architectural requirements of the Coherence-before relation and as a result, this execution is architecturally forbidden.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.5 Reading interrupt configurations and subsequent updates

## B1.5.1 Notes

The purpose of these tests is to show that when reading the configuration of an interrupt and then updating the configuration of that same interrupt, the value read is not affected by the updates.

## B1.5.2 Litmus test with update to priority

```
AArch64 coRW-gic+rcfg-pri
{
 [INTID(A)]=(priority:0);
 0:X0=(intid:A);
 0:X1=(intid:A, priority:1);
}
 P0                    ;
 GIC CDRCFG, X0        ;
 GIC CDPRI, X1         ;
 ISB                   ;
 MRS X3, ICC_ICSR_EL1  ;

exists(0:X3=(priority:1))

kinds: coRW-gic+rcfg-pri Forbid
```

### B1.5.2.1 Explanation

On P0:

- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$.
- `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_2$.

It is architecturally required that the Interrupt Read Effect generated by an `GIC` instruction with the `RCFG` command $E_1$ does not Read-from the Interrupt Write Effect $E_2$.

Note that the `ISB` and `MRS X3, ICC_ICSR_EL1` sequence can be placed before the `GIC CDPRI, X1` instruction without changing the outcome of the test.

## B1.5.3 Litmus test with deactivate

```
AArch64 coRW-gic+rcfg-di
{
 [INTID(A)]=(active:1);
 0:X0=(intid:A);
 0:X1=(intid:A);
}
 P0                    ;
 GIC CDRCFG, X0        ;
 GIC CDDI, X1          ;
 ISB                   ;
 MRS X3, ICC_ICSR_EL1  ;

exists(0:X3=(active:0))

kinds: coRW-gic+rcfg-di Forbid
```

### B1.5.3.1 Explanation

On P0:

- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$.
- `GIC CDDI, X1` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_2$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

It is architecturally required the Interrupt Read Effect generated by an `GIC` instruction with the `RCFG` command $E_1$ does not Read-from the Interrupt Write effect $E_2$.

Note that the `ISB` and `MRS X3, ICC_ICSR_EL1` sequence can be placed before the `GIC CDDI, X1` instruction without changing the outcome of the test.

### B1.5.4 Litmus test with acknowledge

```
AArch64 coRW-gic+rcfg-ia
{
 [INTID(A)]=(priority:1,pending:1);
 0:X0=(intid:A);
}
 P0                    ;
 GIC CDRCFG, X0        ;
 GICR X1,CDIA          ;
 ISB                   ;
 MRS X3, ICC_ICSR_EL1  ;

exists(0:X3=(priority:1,pending:0,active:1))

kinds: coRW-gic+rcfg-ia Forbid
```

#### B1.5.4.1 Explanation

On P0:

- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$.
- `GICR X1,CDIA` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_2$.

It is architecturally required the Interrupt Read Effect generated by an `GIC` instruction with the `RCFG` command $E_1$ does not Read-from the Interrupt Write effect $E_2$.

Note that the `ISB` and `MRS X3, ICC_ICSR_EL1` sequence can be placed before the `GICR X1,CDIA` instruction without changing the outcome of the test.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.6 Configuration and acknowledgement

## B1.6.1 Notes

The purpose of these tests is to show when the effects of configuration updates are required to be observed by a `GICR CDIA` or `GICR CDNMIA` instruction when using explicit synchronization.

Illustrates a common principle of interrupt acknowledgement not being guaranteed to observe a preceding write in program order without either using explicit synchronization or observing the write through an explicit read.

See also:

- B1.11 *IPI and acknowledgement*
- B1.33 *Reading interrupt configuration and exception status*

## B1.6.2 Litmus test using disable without explicit synchronization

```
AArch64 coWR-gic.dis.ack
{
 [INTID(A)]=(enabled:1, pending:1);
 0:X1=(intid:A);
}
 P0                  ;
 GIC  CDDIS, X1      ;
 GICR X0, CDIA       ;

exists(0:X0=(valid:1, intid:A))

kinds: coWR-gic.dis.ack Allow
```

### B1.6.2.1 Explanation

In the execution that satisfies the postcondition, on P0:

- `GIC CDDIS, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GICR X0, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_2$.

In the execution that satisfies the postcondition, $E_2$ is Coherence-before $E_1$. This is permitted by the architecture.

## B1.6.3 Litmus test using disable with explicit synchronization

```
AArch64 coWR-gic.dis.ack+gsb.sys
{
 [INTID(A)]=(enabled:1, pending:1);
 0:X1=(intid:A);
}
 P0                  ;
 GIC  CDDIS, X1      ;
 GSB  SYS            ;
 GICR X0, CDIA       ;

exists(0:X0=(valid:1, intid:A))

kinds: coWR-gic.dis.ack+gsb.sys Forbid
```

### B1.6.3.1 Explanation

In the execution that satisfies the postcondition, on P0:

- `GIC CDDIS, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GSB SYS` generates a GSB.SYS Effect $E_2$.
- `GICR X0, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_3$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

Therefore, $E_1$ is GSB-Ordered-before $E_3$. As per the postcondition, $E_3$ is Coherence-before $E_1$. Consequently, this execution creates a cycle in the Ordered-before relation and as a result, violates the External visibility requirement.

### B1.6.4 Litmus test with deactivate

```
AArch64 coWW-gic+di-ia+rcfg
{
 [INTID(A)]=(pending:1,enabled:1,affinity:P0);
 0:X0=(intid:A);
 1:X0=(intid:A);
}
 P0            | P1                  ;
 GIC CDDI,X0   | GIC CDRCFG,X0       ;
 GICR X1,CDIA  | ISB                 ;
               | MRS X2,ICC_ICSR_EL1 ;
exists(0:X1=(valid:1,intid:A) /\ 1:X2=(pending:0,active:0,enabled:

kinds: coWW-gic+di-ia+rcfg
```

#### B1.6.4.1 Explanation

In the execution that satisfies the postcondition, on P0:

- `GIC CDDI, X0` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_2$.
- `GICR X1, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$.
- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition, $E_4$ is Coherence-before $E_2$ and $E_5$ Reads-from $E_2$. This means that $E_2$ is Coherence-before $E_3$ and that violates the architectural requirements of the Coherence-before relation. As a result, this execution is architecturally forbidden.

### B1.6.5 Litmus test using priority without explicit synchronization

```
AArch64 coWR-gic+pcr-isb-pri-ia
{
 [INTID(A)]=(enabled:1, pending:1, priority:1);
 0:X1=(intid:A, priority:2);
}
 P0                  ;
 MOV X0, #1          ;
 MSR ICC_PCR_EL1, X0 ;
 ISB                 ;
 GIC CDPRI, X1       ;
 GICR X0, CDIA       ;

exists(0:X0=(valid:1, intid:A))

kinds: coWR-gic+pcr-isb-pri-ia Allow
~~~
```

#### B1.6.5.1 Explanation

In the execution that satisfies the postcondition, on P0:

- `MSR ICC_PCR_EL1, X0` generates a Direct System Register Write Effect to ICC_PCR_EL1 $E_1$.
- `ISB` generates a Instruction Fetch Barrier Effect $E_2$.
- `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_3$.
- `GICR X0, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Indirect System Register Read Effect to ICC_PCR_EL1 $E_5$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

$E_1$ is IFB-Ordered-before $E_5$ and, consequently, $E_5$ Reads-from-register $E_1$. This means that `GICR X0, CDIA` cannot acknowledge any interrupt with priority lower than 1. As per the postcondition, `GICR X0, CDIA` acknowledges interrupt `INTID(A)` and since the priority of INTID(A) cannot have been lower than 1, $E_4$ is Coherence-before $E_3$. This is an architecturally allowed execution.

### B1.6.6 Litmus test using priority with explicit synchronization.

```
AArch64 coWR-gic+pcr-isb-pri-gsb.sys-ia
{
 [INTID(A)]=(enabled:1, pending:1, priority:1);
 0:X1=(intid:A, priority:2);
}
 P0                   ;
 MOV  X0, #1          ;
 MSR  ICC_PCR_EL1, X0 ;
 ISB                  ;
 GIC  CDPRI, X1       ;
 GSB  SYS             ;
 GICR X0, CDIA        ;

exists(0:X0=(valid:1, intid:A))

kinds: coWR-gic+pcr-isb-pri-gsb.sys-ia Forbid
```

#### B1.6.6.1 Explanation

In the execution that satisfies the postcondition on P0:

- `MSR ICC_PCR_EL1, X0` generates a Direct System Register Write Effect to ICC_PCR_EL1 $E_1$.
- `ISB` generates a Instruction Fetch Barrier Effect $E_2$.
- `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_3$.
- `GSB SYS` generates a GSB.SYS Effect $E_4$.
- `GICR X0, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Indirect System Register Read Effect to ICC_PCR_EL1 $E_6$.

$E_1$ is IFB-Ordered-before $E_2$ and, consequently, $E_6$ Reads-from-register $E_1$. This means that `GICR X0, CDIA` cannot acknowledge any interrupt with priority lower than 1. As per the postcondition, `GICR X0, CDIA` acknowledges interrupt `INTID(A)` and since the priority of INTID(A) cannot have been lower than 1, $E_5$ is Coherence-before $E_3$. At the same time $E_3$ is GSB-Ordered-before $E_5$. This creates a cycle in the Ordered-before relation which violates the external visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.7 Acknowledge followed by interrupt changes

## B1.7.1 Notes

The update to an interrupt caused by an interrupt acknowledge is required to be ordered with respect to a deactivate but not with any other configuration update.

## B1.7.2 Litmus test with deactivate

```
AArch64 ack-deactivate
{
 [INTID(A)]=(enabled:1,affinity:P0,pending:1,active:0);
 0:X0=(intid:A);
}
 P0                      ;
 GICR  X1, CDIA          ;
 GIC   CDDI, X0          ;
 GIC   CDRCFG, X0        ;
 ISB                     ;
 MRS   X2, ICC_ICSR_EL1  ;

exists(0:X1=(valid:1) /\ 0:X2=(pending:0, active:1))

kinds: ack-deactivate Forbid
```

### B1.7.2.1 Explanation

In the execution that satisfies the postcondition on P

- `GICR X1, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_1$. As per the postcondition, the `GICR` instruction acknowledges `INTID(A)` and, as a result, generates an Implicit Interrupt Write Effect $E_2$ to `INTID(A)` that resets its pending state to 0 and sets its active state to 1.
- `GIC CDDI, X0` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_3$.
- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_4$.

As per the postcondition, the Explicit Interrupt Read Effect $E_4$ observes that `INTID(A)` is active, and as a result, $E_4$ Reads-from $E_2$. This means that either:

- $E_4$ is Coherence-before relation $E_3$ which violates the architectural requirements of the Coherence-before relation, or
- $E_3$ is Coherence-before relation $E_2$ and concequently $E_1$ Reads-from $E_3$ which violates the architectural requirements of the Reads-from relation.

## B1.7.3 Litmus test with make pending

```
AArch64 edge-merging.ipi
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P0);
 0:X0=(intid:A, pending:1); 0:X1=(intid:A);
}
 P0                       ;
 GICR    X2, CDIA         ;
 GIC     CDPEND, X0       ;
 GIC     CDRCFG, X1       ;
 ISB                      ;
 MRS     X3, ICC_ICSR_EL1 ;

exists(0:X2=(valid:1, intid:A) /\
       0:X3=(pending:0, active:1))

kinds: edge-merging.ipi Forbid
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### B1.7.3.1 Explanation

On P0:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_2$ that resets the pending state to 0 and sets the active state to 1.
- `GIC CDPEND,X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ that sets the pending state to 1.
- `GIC CDRCFG,X1` generates an Explicit Interrupt Read Effect $E_5$ to `INTID(A)`.

In the execution that satisfies the postcondition, $E_5$ Reads-from $E_2$, and any of the following applies:

- $E_4$ is Coherence-before $E_2$. This violates the requirements of the Coherence-before and, as a result, is architecturally forbidden.
- $E_2$ is Coherence-before $E_4$ and, consequently, $E_5$ is Coherence-before $E_4$. This violates the requirements of the Coherence-before and, as a result, is architecturally forbidden.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.8 Multiple updates with interleaved read

## B1.8.1 Notes

The purpose of this test is to show that a write, followed by a read, followed by a write to the same interrupt location must result in the read observing the first write and not the second write.

## B1.8.2 Litmus test

```
AArch64 coWRW-gic
{
 [INTID(A)]=(priority:0);
 0:X0=(intid:A);
 0:X1=(intid:A, priority:1);
 0:X2=(intid:A, priority:2);
}
 P0                    ;
 GIC CDPRI, X1         ;
 GIC CDRCFG, X0        ;
 GIC CDPRI, X2         ;
 ISB                   ;
 MRS X3, ICC_ICSR_EL1  ;

exists(0:X3=(priority:2) \/
       0:X3=(priority:0))

kinds: coWRW-gic Forbid
```

### B1.8.2.1 Explanation

On P0:

- The first `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect $E_1$.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect $E_2$.
- The second `GIC CDPRI, X2` generates an Explicit Interrupt Write Effect $E_3$.

It is architecturally required that:

- The Interrupt Write Effect generated by an `GIC` instruction with the `RCFG` command $E_2$ is Coherence-after the Interrupt Write effect $E_1$.
- The Interrupt Read Effect generated by an `GIC` instruction with the `RCFG` command $E_1$ does not Read-from the Interrupt Write Effect $E_3$.

Note that the `ISB` and `MRS X3, ICC_ICSR_EL1` sequence can be placed before the second `GIC CDPRI, X2` instruction without changing the outcome of the test.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.9 Configuration write and IRQ unmask in PSTATE

## B1.9.1 Notes

The purpose of these tests is to show the required synchronization of to ensure that after disabling an pending interrupt there cannot be a IRQ exception when possible.

See 2.6.1 *GIC synchronization barriers (GSBs)* and GSB-ordered-before for more information related to the litmus tests in this section.

## B1.9.2 Litmus test

```
AArch64 gic.cddis-gsb.sys-mrs.isr_el1
{
 [INTID(A)]=(enabled:1, pending:1);
 0:X1=(intid:A);
}
 P0               ;
 GIC CDDIS, X1    ;
 GSB SYS          ;
 MRS  X0, ISR_EL1 ;
exists(0:X0=(I=1))

kinds: gic.cddis-gsb.sys-mrs.isr_el1 Forbid
```

### B1.9.2.1 Explanation

For the execution that validates the postcondition:

- `GIC CDDIS,X1` generates an Interrupt Write Effect $E_1$ to Interrupt Location `INTID(A)`.
- `GSB SYS` generates a GSB.SYS Effect $E_2$.
- `MRS X0, ISR_EL1` generates a Direct System Register Read to `ISR_EL1` $E_3$.

In the execution that satisfies the postcondition, $E_3$ indicates that `INTID(A)` is still the HPPI. However because of $E_2$ `INTID(A)` is disabled and cannot be the HPPI. As a result, this execution is architecturally forbidden.

## B1.9.3 Litmus test with initially masked IRQs

```
AArch64 gic.cddis-gsb.sys-clr.i
{
 [INTID(A)]=(enabled:1, pending:1);
 0:X1=(intid:A);
}
 P0                ;
 GIC  CDDIS  X1    ;
 GSB  SYS          ;
 DAIFClr  PSTATE.I ;

exists(async(P0, IRQ))

kinds: coWex+gic.cddis-gsb.sys-clr.i Forbid
```

### B1.9.3.1 Explanation

For the execution that validates the postcondition:

- `GIC CDDIS,X1` generates an Interrupt Write Effect $E_1$ to Interrupt Location `INTID(A)`.
- `GSB SYS` generates a GSB.SYS Effect $E_2$.
- `DAIFClr PSTATE.I` generates a Direct System Register Write to `PSTATE` $E_3$.

In the execution that satisfies the postcondition, `INTID(A)` has become a pending IRQ. This execution is architectural forbidden as the `GSB.SYS` is required to have made the $E_1$ observable before the IRQ was unmasked.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### B1.9.4 Litmus test with disable of a PPI

```
AArch64 ppi-disable-isb-clr.i
{
 0:[ppi(27)]=(enabled:1,pending:1,active:0);
}
 P0
 MOV   X0, #0                    ;
 MSR   ICC_PPI_ENABLER, X0       ;
 ISB                             ;
 DAIFClr  PSTATE.I               ;

 exists(async(P0, IRQ)) Forbid
```

#### B1.9.4.1 Explanation

The ISB guarantees completion of the MSR instructions and the effects of disabling the PPI which can no longer be the HPPI. As a result, this execution is architecturally forbidden.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.10 Configuration write and exception status on a single PE

## B1.10.1 Notes

The purpose of this test is to show that an interrupt exception as a result of enabling a pending interrupt is not required to be observed even when explicit synchronization is used.

## B1.10.2 Litmus test without wait for IRQ exception to be signaled

```
AArch64 coWR+gic.cden-gsb.sys-isb-isr
{
 [INTID(A)]=(enabled:0, pending:1);
 0:X1=(intid:A);
}
 P0                    ;
 GIC  CDEN, X1         ;
 GSB  SYS              ;
 ISB                   ;
 MRS  X0, ISR_EL1      ;

exists(0:X0=(I=0))

kinds: coWR+gic.cden-gsb.sys-isb-isr Allow
```

### B1.10.2.1 Explanation

On P0:

- `GIC CDEN, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GSB SYS` generates a GSB.SYS Effect $E_2$.
- `ISB` generates a Instruction Fetch Barrier Effect $E_3$.
- `MRS X0, ISR_EL1` generates a Direct System Register Read Effect to ISR_EL1 $E_4$.

$E_1$ is IFB-Ordered-before $E_1$. As a result, `INTID(A)` is already a candidate HPPI when P0 generates $E_4$. However, the architecture does not guarantee that INTID(A) will be presented as the HPPI immediately. And as a result, the execution where reading ISR_EL1 indicatates there is no IRQ pending is architecturally allowed.

## B1.10.3 Litmus test with wait for IRQ exception to be signaled

```
AArch64 coWR+gic.cden-gsb.sys-isb-isr+wait
{
 [INTID(A)]=(enabled:0, pending:1);
 0:X1=(intid:A);
}
 P0                                ;
      GIC  CDEN, X1                ;
      GSB  SYS                     ;
      ISB                          ;
 WAIT (MRS  X0, ISR_EL1; X0=(I=1))  ;

exists(0:X0=(I=0))

kinds: coWR+gic.cden-gsb.sys-isb-isr+wait Forbid
```

### B1.10.3.1 Explanation

On P0:

- `GIC CDEN, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GSB SYS` generates a GSB.SYS Effect $E_2$.
- `ISB` generates a Instruction Fetch Barrier Effect $E_3$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- The execution of `MRS X0, ISR_EL1` in a loop generates a Direct System Register Read Effect to ISR_EL1 $E_4$, $E_5$, ..., $E_n$.

$E_1$ is IFB-Ordered-before $E_4$. As a result, `INTID(A)` is already a candidate HPPI when P0 generates $E_4$. The architecture does not guarantee that INTID(A) will be presented as the HPPI immediately. However, in finite time the GIC will presented `INTID(A)` as the HPPI, as a result, $E_n$ where n is a finite number reads that the `I` field of ISR_EL1 is set to 1 and an IRQ is pending.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.11 IPI and acknowledgement

## B1.11.1 Notes

The purpose of these tests is to show that sending an IPI is only required to be observed by an acknowledge when using explicit synchronization.

Illustrates a common principle of interrupt acknowledgement not being guaranteed to observe a preceding write in program order without either using explicit synchronization or observing the write through an explicit read.

See also:

- B1.6 *Configuration and acknowledgement*

## B1.11.2 Litmus test without explicit synchronization

```
AArch64 coWR-gic.ipi.ack
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P0);
 0:X0=(intid:A, pending:1); 0:X1=(intid:A);
}
 P0                        ;
 GIC     CDPEND, X0        ;
 GICR    X2, CDIA          ;
 GIC     CDRCFG, X1        ;
 ISB                       ;
 MRS     X3, ICC_ICSR_EL1  ;

exists(0:X2=(valid:1) /\ 0:X3=(pending:1, [illegible]:1))

kinds: coWR-gic.ipi.ack Allow
```

### B1.11.2.1 Explanation

On P0:

- `GIC CDPEND, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets the `pending` state to 1.
- `GICR X2, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$ that sets the `pending` state to 0 and sets the `active` state to 1.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_6$.

In the execution that satisfies the postcondition, `GICR X2, CDIA` acknowledges `INTID(A)` and consecuently, either:

- $E_6$ Reads-from the initial value of `INTID(A)` and therefore $E_6$ is Coherence-before $E_2$. This violates architectural requirements of the Coherence-before relation.
- $E_6$ Reads-from $E_2$. This means that either:
  - $E_2$ is Coherence-before $E_5$ and $E_6$ is Coherence-before $E_5$. This violates architectural requirements of the Coherence-before relation.
  - $E_5$ is Coherence-before $E_2$. This execution is architecturally allowed.

## B1.11.3 Litmus test with explicit synchronization

```
AArch64 coWR-gic.ipi.ack+gsb.sys
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P0);
 0:X0=(intid:A, pending:1); 0:X1=(intid:A);
}
 P0                        ;
 GIC     CDPEND, X0        ;
 GSB     SYS               ;
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

```
 GICR    X2, CDIA          ;
 GIC     CDRCFG, X1        ;
 ISB                       ;
 MRS     X3, ICC_ICSR_EL1  ;

exists(0:X2=(valid:1) /\ 0:X3=(pending:1))

kinds: coWR-gic.ipi.ack+gsb.sys Forbid
```

### B1.11.3.1 Explanation

On P0:

- `GIC CDPEND, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets the `pending` state to 1.
- `GSB SYS` generates a GSB.SYS Effect $E_3$.
- `GICR X2, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$ that resets the `pending` state to 0 and sets the `active` state to 1.
- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_6$.

In the execution that satisfies the postcondition, `GICR X2, CDIA` acknowledges `INTID(A)` and consecuently, either:

- $E_6$ Reads-from the initial value of `INTID(A)` and therefore $E_6$ is Coherence-before $E_2$. This violates architectural requirements of the Coherence-before relation.
- $E_6$ Reads-from $E_2$. This means that either:
  - $E_2$ is Coherence-before $E_5$ and $E_6$ is Coherence-before $E_5$. This violates architectural requirements of the Coherence-before relation.
  - $E_5$ is Coherence-before $E_2$ which would mean that $E_1$ Reads-from $E_5$ and as a result $E_5$ is GIC-Observed-by $P_1$. At the same time, $E_1$ is GSB-ordered-before $E_5$. This creates a cycle in the Ordered-before relation and as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.12 Observing multiple writes on a different PE

## B1.12.1 Notes

The purpose of this test is to show that the external observations of updates to the same interrupt location cannot contradict the Coherence order.

## B1.12.2 Litmus test

```
AArch64 coWW-gic+R
{
 [INTID(A)]=(enabled:0, affinity:P0);
 0:X1=(intid:A, affinity:P1);
 0:X2=(intid:A);
 1:X1=(intid:A);
}
 P0               | P1                  ;
 GIC CDAFF, X1    | GIC CDRCFG, X1      ;
 GIC CDEN, X2     | ISB                 ;
                  | MRS X0, ICC_ICSR_EL1 ;

exists(1:X0=(affinity:P0, enabled:1))

kinds: coWW-gic+R Forbid
```

### B1.12.2.1 Explanation

On P0:

- `GIC CDAFF, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- `GIC CDEN, X2` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_2$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_3$.

On P1:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_4$.

For the execution that satisfies the post-condition, $E_4$ Reads-from $E_3$ and $E_2$ is Coherence-before $E_1$. This violates the architectural requirement of the Reads-from relation.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.13 Read of the configuration of an interrupt

## B1.13.1 Notes

The purpose of this section if to demonstrate the ordering properties of the Indirect System Register Write to ICC_ICSR_EL1 generated by the execution of a GIC system instruction with the `RCFG` command.

## B1.13.2 Litmus test without ISB

```
AArch64 gic.rcfg-mrs.icc_icsr_el1
{
 [INTID(A)]=(priority:5);
 0:X0=(intid:A);
}
 P0                  ;
 GIC CDRCFG, X0      ;
 MRS X1, ICC_ICSR_EL1 ;

exists(1:X1!=(priority:5))

kinds: gic.rcfg-mrs.icc_icsr_el1 Allow
```

### B1.13.2.1 Explanation

On P0:

- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect $E_1$ and an Indirect System Register Write Effect $E_2$.
- `MRS X1, ICC_ICSR_EL1` generates a Direct System Register Read Effect to ICC_ICSR_EL1 $E_3$ and a Register Write Effect to `X1` $E_4$.

In the execution that satisfies the postcondition $E_1$ Reads-from the initial configuration of `INTID(A)` and consequently $E_3$ is Coherence-before $E_1$. This is architecturally allowed.

## B1.13.3 Litmus test with ISB

```
AArch64 gic.rcfg-isb-mrs.icc_icsr_el1
{
 [INTID(A)]=(priority:5);
 0:X0=(intid:A);
}
 P0                  ;
 GIC CDRCFG, X0      ;
 ISB                 ;
 MRS X1, ICC_ICSR_EL1 ;

exists(1:X1!=(priority:5))

kinds: gic.rcfg-isb-mrs.icc_icsr_el1 Forbid
```

### B1.13.3.1 Explanation

On P0:

- `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect $E_1$ and an Indirect System Register Write Effect $E_2$.
- `ISB` generates an Instruction Fetch Barrier Effect $E_2$ which is a Context Synchronization Event.
- `MRS X1, ICC_ICSR_EL1` generates a Direct System Register Read Effect to ICC_ICSR_EL1 $E_3$ and a Register Write Effect to `X1` $E_4$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

In the execution that satisfies the postcondition, $E_1$ Reads-from the initial configuration of `INTID(A)` and consequently $E_3$ is Coherence-before $E_1$. This is architecturally forbidden. $E_3$ is required to Read-from $E_1$ because $E_2$ is a Context Synchronization Event.

See also *Arm® Architecture Reference Manual, for A-profile architecture*[1] Table D24-1 Synchronization requirements' for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.14 Multiple reads of the same config

## B1.14.1 Notes

The purpose of this test is to show that if a read has observed a write to the same location, a second read must also have observed the same write.

The first litmus test uses an ISB after both `GIC CDRCFG, X0` instructions to ensure that the implicit writes to ICC_ICSR_EL1 are observed by each `MRS` instruction after the ISB. The subsequent litmus tests illustrate the impact of omitting one or both ISBs.

## B1.14.2 Litmus test with ISBs

```
AArch64 coRR-gic+cdpri+cfg-isb-cfg-isb
{
 [INTID(A)]=(priority:0);
 0:X1=(intid:A, priority:1);
 1:X0=(intid:A);
}
 P0            | P1                  ;
 GIC CDPRI, X1 | GIC CDRCFG, X0       ;
               | ISB                 ;
               | MRS X1, ICC_ICSR_EL1 ;
               | GIC CDRCFG, X0       ;
               | ISB                 ;
               | MRS X2, ICC_ICSR_EL1 ;

exists(1:X1=(priority:1) /\ 1:X2=(priori  0))

kinds: coRR-gic+cdpri+cfg-isb-cfg-is  orbid
```

### B1.14.2.1 Explanation

On P0:

- `GIC CDPRI, X1` nerates an Expli Interrupt Write Effect $E_1$.

On P1:

- The first `GIC  G, X0` ge rates an Explicit Interrupt Read Effect $E_2$ and an Indirect System Register Writ ct to I ICSR L1 $E_3$.
- he first I B generate n IFB Effect $E_4$.
- R C ICC_I EL1 generates a Direct System Register Read Effect E~5.
- The ond `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect $E_6$ and an Indirect System Register Write E ct to ICC_ICSR_EL1 $E_7$.
- The second `ISB` generates an IFB Effect $E_8$.
- `MSR C1, ICC_ICSR_EL1` generates a Direct System Register Read Effect E~9.
- The first `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect $E_{10}$ and an Indirect Write to System Register ICC_ICSR_EL1 $E_{11}$.

In the execution that satisfies the postcondition, $E_2$ Reads-from $E_1$ and as a result $E_2$, $E_1$ is Observer-by $E_2$ and one of the following applies:

$E_3$ is Coherence-before $E_1$ and as a result, $E_2$ is GIC-Hazard-Ordered-before $E_1$. This creates a cycle in the Ordered-before relation which violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.15 GIC coherence order

## B1.15.1 Notes

The purpose of this test is to show that due to the total order on writes to an interrupt location, two reads of the same location by two separate observers must observe the same order on the writes.

## B1.15.2 Litmus test

```
AArch64 IRIW-loc+gic.cdpri
{
 [INTID(A)]=(priority:0);
 0:X1=(intid:A, priority:1);
 1:X1=(intid:A, priority:2);
 2:X0=(intid:A);
 3:X0=(intid:A);
}
 P0            | P1            | P2                   | P3
 GIC CDPRI, X1 | GIC CDPRI, X1 | GIC CDRCFG, X0       | GIC CDRCFG, X0
               |               | ISB                  | ISB                  ;
               |               | MRS X1, ICC_ICSR_EL1 | MRS X1, ICC_ICSR_EL1 ;
               |               | GIC CDRCFG, X0       | GIC CDRCFG, X0       ;
               |               | ISB                  | ISB                  ;
               |               | MRS X2, ICC_ICSR_EL1 | MRS X2, ICC_ICSR_EL1 ;

exists((2:X1=(priority:1) /\ 2:X2=(priority:2) /\
        3:X1=(priority:2) /\ 3:X2=(priority:1))

kinds: IRIW-loc+gic.cdpri Forbid
```

### B1.15.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0, `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ that sets its priority to 1.
- On P1, `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets its priority to 2.
- On P2:
  - The first `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ that reads the priority as 1.
  - The second `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_4$ that reads the priority as
- On P3:
  - The first `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$ that reads the priority as 2.
  - The second `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_6$ that reads the priority as 1.

Any of the following applies:

- $E_1$ is Coherence-before $E_2$. In which case $E_6$ is Coherence-before $E_2$ and consecuently $E_5$ is GIC-hazard-ordered-before $E_2$. At the same time, $E_5$ [Reads-from](#def:cpuif:ordering_model:rf] $E_2$ and as a result $E_2$ is [GIC-observed-by](#def:cpuif:ordering_model:GIC-obs] $E_5$. This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.
- $E_2$ is Coherence-before $E_1$. In which case $E_4$ is Coherence-before $E_1$ and consecuently $E_3$ is GIC-hazard-ordered-before $E_1$. At the same time, $E_3$ [Reads-from](#def:cpuif:ordering_model:rf] $E_1$ and as a result $E_1$ is [GIC-observed-by](#def:cpuif:ordering_model:GIC-obs] $E_3$. This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.16 Independent reads of independent writes

## B1.16.1 Notes

The purpose of this test is to show that writes which can be observed by any observer must be observable by all observers.

In other words, the test shows that the interrupt ordering model has other multicopy atomicity.

## B1.16.2 Litmus test

```
AArch64 IRIW-2loc+gic.cdpri
{
 [INTID(A)]=(priority:0);
 [INTID(B)]=(priority:0);
 0:X1=(intid:A, priority:1);
 1:X1=(intid:B, priority:1);
 2:X0=(intid:A); 2:X1=(intid:B);
 3:X0=(intid:A); 3:X1=(intid:B);
}
 P0            | P1            | P2                      | P3                      ;
 GIC CDPRI, X1 | GIC CDPRI, X1 | GIC CDRCFG, X0          | GIC CDRCFG, X1          ;
               |               | ISB                     | ISB                     ;
               |               | MRS X2, ICC_ICSR_EL1    | MRS X2, ICC_ICSR_EL1    ;
               |               | GSB SYS                 | GSB SYS                 ;
               |               | GIC CDRCFG, X1          | GIC CDRCFG, X0          ;
               |               | ISB                     | ISB                     ;
               |               | MRS X3, ICC_ICSR_EL1    | MRS X3, ICC_ICSR_EL1    ;

exists(2:X2=(priority:1) /\ 2:X3=(priority:0) /\
       3:X2=(priority:1) /\ 3:X3=(priority:0))

kinds: IRIW-2loc+gic.cdpri Forbid
```

### B1.16.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0, `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$.
- On P1, `GIC CDPRI, X1` generates an Explicit Interrupt Write Effect to `INTID(B)` $E_2$.
- On P2:
  - `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$.
  - `GSB SYS` generates a GSB SYS Effect $E_4$.
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(B)` $E_5$.
- On P3:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(B)` $E_6$.
  - `GSB SYS` generates a GSB SYS Effect $E_7$.
  - `GIC CDRCFG, X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_8$.

$E_1$ is GIC-observed-by $E_3$ as $E_3$ Reads-from $E_1$. $E_3$ is GSB-ordered-before $E_5$. $E_5$ is GIC-observed-by $E_2$ as $E_5$ is Coherence-before $E_2$. $E_2$ is GIC-observed-by $E_6$ as $E_6$ Reads-from $E_2$. $E_6$ is GSB-ordered-before $E_8$. $E_8$ is GIC-observed-by $E_1$ as $E_8$ is Coherence-before $E_1$. As a result, this execution creates a cycle in the Ordered-before relation and violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.17 Message passing via flag in memory

## B1.17.1 Notes

The purpose of this test is to show the required barrier use when using a flag in memory to inform a different PE that an interrupt configuration has been updated.

## B1.17.2 Litmus test

```
AArch64 MP+gsb.sys+dsb.ld
{
 [INTID(A)]=(affinity:P0);
 0:X1=x;
 0:X2=(intid:A, affinity:P1);
 0:X3=(intid:A);
 1:X1=x;
 1:X2=(intid:A);
}
 P0              | P1                    ;
 GIC  CDAFF, X2  | LDR  X0, [X1]         ;
 GSB  SYS        | DSB  LD               ;
 MOV  X0, #1     | GIC  CDRCFG, X2       ;
 STR  X0, [X1]   | ISB                   ;
                 | MRS  X3, ICC_ICSR_EL1 ;

exists(1:X0=1 /\ 1:X3=(affinity:P0))

kinds: MP+gsb.sys+dsb.ld Forbid
```

### B1.17.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `GIC CDAFF, X2` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ that sets its affinity to P1.
  - `GSB SYS` generates a GSB SYS Effect $E_2$.
  - `STR X0, [X1]` generates an Explicit Write Memory Effect to `x` $E_3$.
- On P1:
  - `LDR X0, [X1]` generates an Explicit Memory Read Effect to `x` $E_4$.
  - `DSB LD` generates a DSB LD Effect $E_5$.
  - `GIC RCFG, X2` generates an Explicit Interrupt Effect $E_6$.

In this execution:

- $E_4$ Reads-from-memory $E_3$ and as a result $E_3$ is Explicit-observed-by $E_4$.
- $E_4$ is DSB-ordered-before $E_6$.
- $E_6$ is Coherence-before $E_1$.
- $E_1$ is GSB-ordered-before $E_4$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.18 Message passing via interrupt priority configuration

## B1.18.1 Notes

The purpose of this test is to show that writes to an interrupt configuration can be ordered with respect to an update of the interrupt Pending state such that observing the interrupt as Pending guarantees that the writes to the interrupt configuration are observed.

## B1.18.2 Litmus test

```
AArch64 MP-pri+gsb.sys+ack
{
 [INTID(A)]=(enabled:1,affinity:P1,priority:1,pending:0);
 [PTE(x)]=(oa:PA(x),attrs:(device-nGRE)); // PERIP
 0:X1=x;
 0:X2=(intid:A, priority:5);
 1:X1=(intid:A);
}
 P0              | P1                    ;
 GIC CDPRI, X2   | GICR X0, CDIA         ;
 GSB SYS         | GIC  CDRCFG, X1       ;
 STR X0, [X1]    | ISB                   ;
                 | MRS  X1, ICC_ICSR_EL1 ;

exists(1:X0=(valid:1, intid:A) /\
       1:X1=(enabled:1,affinity:P1,priority:1,active:1))

kinds: MP-pri+gsb.sys+ack Forbid
```

### B1.18.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `GIC CDPRI, X2` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_1$ that sets its priority to 5.
  - `GSB SYS` generates a GSB SYS Effect $E_2$.
  - `STR X0, [X1]` generates an Explicit Write Memory Effect to `x`. When the Explicit Write Effect to location `x` reaches its endpoint, the Peripheral device signals an interrupt event to the IRI which generates an Interrupt Read Effect $E_3$ to `INTID(A)` and an Interrupt Write Effect to `INTID(A)` $E_4$ which sets the pending state to 1.
- On P1:
  - `GICR X0,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_6$ that resets the pending state to 0 and sets the active state to 1.
  - `GIC CDRCFG, X2` generates an Explicit Interrupt Read Effect $E_7$.

In this execution:

- $E_7$ Reads-from $E_6$ and as a result $E_6$ and $E_5$ are Coherence-before $E_1$.
- $E_1$ is GSB-ordered-before $E_4$.
- $E_5$ Reads-from $E_4$ and as a result, $E_4$ is GIC-observed-by $E_5$

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.19 Message passing with an LPI and a device read

## B1.19.1 Notes

The purpose of this test is to show that if a write to a peripheral has the side effect of making an interrupt pending, then observing that pending state on a different PE means that the write to the peripheral can also be observed.

## B1.19.2 Litmus test

```
AArch64  MP-fLPI+imp+gsb.ack
{
 [INTID(A)]=(enabled:1,affinity:P1,priority:1,pending:0);
 [PTE(x)]=(oa:PA(x),attrs:(device-nGRE)); // PERIP
 0:X3=x;
 1:X1=x;
}
 P0              | P1              ;
 MOV  W2, #1     | GICR X0, CDIA   ;
 STR W2, [X3]    | GSB  ACK        ;
                 | LDR  W2, [X1]   ;

exists(1:X0=(valid:1, intid:A) /\ 1:X2=0);

kinds: MP-fLPI+imp+gsb.ack Forbid
```

### B1.19.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR W2,[X3]` generates an Explicit Write Memory Effect to `x` $E_1$. When the Explicit Write Effect to Location `x` reaches its endpoint, the Peripheral device signals an interrupt event to the IRI which generates Interrupt Read Effect $E_2$ to `INTID(A)` and an Interrupt Write Effect to `INTID(A)` $E_3$ which sets the pending state to 1.
- On P1:
  - `GICR X0, CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$ that resets the pending state to 0 and sets the active state to 1.

    `GSB ACK` generates a GSB SYS Effect $E_6$.
  - `LDR W2, [X1]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_4$ Reads-from $E_3$ and as a result $E_3$ is GIC-observed-by $E_4$
- $E_4$ is GSB-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and as a result $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is Ordered-before $E_3$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

## B1.19.3 Litmus test with address dependency

The GSB ACK can be omitted in favor of creating an address dependency on the result of the acknowledge.

This may provide performance benefits in some cases, however, an address dependency cannot always be used a replacement for the `GSB ACK`. For example, a store following `LDR X2, [X1, X5]` without any dependency on

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

previous instructions can be re-ordered before the `GICR X0, CDIA` and the effects of the store can be observed before the effects of the `GICR X0, CDIA` on another PE.

```
AArch64  MP-fLPI+imp+addrpt
{
 [INTID(A)]=(enabled:1,affinity:P1,priority:1,pending:0);
 [PTE(x)]=(oa:PA(x),attrs:(device-nGRE)); // PERIP
 0:X1=x;
 1:X1=x;
}
 P0              | P1                ;
 MOV  W2, #1     | GICR X0, CDIA     ;
 STR  W2, [X1]   | EOR  X5, X0, X0   ;
                 | LDR  W2, [X1, X5] ;

exists(1:X0=(valid:1, intid:A) /\ 1:X2=0);

kinds: MP-fLPI+imp+addrpt Forbid
```

### B1.19.3.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR W2,[X1]` generates an Explicit Write Memory Effect to $x_1$. When the Explicit Write Effect to Location `x` reaches its endpoint, the peripheral device signals an interrupt event to the IRI which generates an Interrupt Read Effect $E_2$ to `INTID(A)` and an Interrupt Write Effect to `INTID(A)` $E_3$ which sets the pending state to 1.
- On P1:
  - `GICR X0,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$ that resets the pending state to 0 and sets the active state to 1.
  - `LDR W2,[X1, X5]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_4$ Reads-from $E_3$ and as a result $E_3$ is GIC-observed-by $E_4$
- There is an address dependency from $E_4$ to $E_7$ and, as a result, $E_4$ is Dependency-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and as a result $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is Ordered-before $E_3$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# B1.20 Message passing with an LPI and a GSB

## B1.20.1 Notes

The purpose of this test is to show that when a write to memory is ordered before a write which has the side effect of making an interrupt pending, then observing the interrupt being pending through acknowledgment on a different PE guarantees that the write is also observed.

If the peripheral generating an interrupt exists in the shared address space of P0 and P1, a the `DSB ST` on P0 can be relaxed to a `DMB ST`, but because the test makes no such assumptions, we must use a `DSB ST`.

## B1.20.2 Litmus test

```
AArch64 MP-fSPI+dsb.st+gsb.ack
{
 [INTID(A)]=(enabled:1,affinity:P1,priority:1,pending:0);
 [PTE(y)]=(oa:PA(y),attrs:(device-nGRE)); // PERIP
 0:X1=x; 0:X3=y;
 1:X1=x;
}
 P0              | P1              ;
 MOV  W0, #1     | GICR X0, CDIA   ;
 STR  W0, [X1]   | GSB  ACK        ;
 DSB  ST         | LDR  X2, [X1]   ;
 MOV  W2, #1     |                 ;
 STR  W2, [X3]   |                 ;

exists(1:X0=(valid:1, intid:A) /\ 1:X2=0)

kinds: MP-fSPI+dsb.st+gsb.ack Forbid
```

### B1.20.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR W0,[X1]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` generates a DSB ST Effect $E_2$.
  - `STR W2,[X3]` generates an Explicit Write Memory Effect to `x`. When the Explicit Write Effect to Location `x` reaches its endpoint, the Peripheral device signals an interrupt event to the IRI which generates an Interrupt Read Effect $E_3$ to `INTID(A)` and an Interrupt Write Effect to `INTID(A)` $E_4$ which sets the pending state to 1.
- On P1:
  - `GICR X0,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_6$ that resets the pending state to 0 and sets the active state to 1.
  - `GSB ACK` generates a GSB ACK Effect $E_7$.
  - `LDR X2,[X1]` generates an Explicit Memory Read Effect $E_8$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_5$ is GSB-ordered-before $E_8$.
- $E_8$ is Coherence-before $E_1$ and, as a result, $E_8$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before $E_4$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.21 Message passing with an IPI and a GSB

## B1.21.1 Notes

The purpose of this test is to show that a write to memory ordered before sending an IPI guarantees that the memory write is observed when the IPI is observed on another PE.

This is illustrating a commonly used pattern for cross-CPU communication by software.

## B1.21.2 Litmus test

```
AArch64 MP-fIPI+dsb.st+gsb.ack
{
 [INTID(A)]=(enabled:1, pending:0, affinity:P1);
 0:X1=x; 0:X2=(intid:A, pending:1);
 1:X1=x;
}
 P0              | P1                ;
 MOV W0, #1      | GICR X0, CDIA     ;
 STR W0, [X1]    | GSB  ACK          ;
 DSB ST          | LDR  W2, [X1]     ;
 GIC CDPEND, X2  |                   ;

exists(1:X0=(valid:1, intid:A) /\ 1:X2=0)

kinds: MP-fIPI+dsb.st+gsb.ack Forbid
```

### B1.21.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR W0,[X1]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` generates a DSB ST Effect $E_2$.
  - `GIC CDPEND, X2` generates an Explicit Interrupt Read Effect $E_3$ to `INTID(A)` and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ which sets the pending state to 1.
- On P1:
  - `GICR X0,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_6$ that resets the pending state to 0 and sets the active state to 1.
  - `GSB ACK` generates a GSB ACK Effect $E_7$.
  - `LDR X2,[X1]` generates an Explicit Memory Read Effect $E_8$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_5$ is GSB-ordered-before $E_8$.
- $E_8$ is Coherence-before $E_1$ and, as a result, $E_8$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before $E_4$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.22 Message passing using deactivate

## B1.22.1 Notes

The purpose of these tests is to show that without explicit synchronization, or a combination of explicit synchronization and an address dependency, there is no ordering between writing to memory and deactivating an interrupt, or between reading interrupt configuration and reading from memory.

This test illustrates part of a real-world example in the following way. P0 illustrates the end of an interrupt handler sequence, which performs memory accesses as part of the handler logic and then finishes the handler by deactivating the interrupt. P1 reads the interrupt configuration and state and determines that the interrupt is no longer actively being handled, and therefore P1 expects to observe all memory accesses performed as part of the handler on P0.

## B1.22.2 Litmus test with explicit synchronization

```
AArch64 MP-fDI+W-dsb-W+R-gsb-R
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A);
}
 P0                    | P1                    ;
 MOV     X3, #1        | GIC CDRCFG, X1        ;
 STR     X3, [X0]      | ISB                   ;
 DSB     ST            | MRS X2, ICC_ICSR_EL1  ;
 GIC     CDDI, X1      | GSB SYS               ;
                       | LDR X3, [X0]

exists(1:X2=(active:0) /\ 1:X3=

kinds: MP-fDI+W-dsb-W+R-gsb  Forbid
```

### B1.22.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR X3,[X0]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` generates a DSB ST Effect $E_2$.
  - `GIC CDDI, X1` generates an Implict Interrupt Read Effect $E_3$ to `INTID(A)` and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$ which resets the active state to 0.
- On P1:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.
  - `GSB SYS` generates a GSB ACK Effect $E_6$.
  - `LDR X3,[X0]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_5$ is GSB-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before $E_4$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### B1.22.3 Litmus test with address dependency

```
AArch64 MP-fDI+W-dsb-W+R-addrpt-R
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A);
}
 P0                  | P1                    ;
 MOV     X3, #1      | GIC CDRCFG, X1        ;
 STR     X3, [X0]    | ISB                   ;
 DSB     ST          | MRS X2, ICC_ICSR_EL1  ;
 GIC     CDDI, X1    | EOR X5, X2, X2        ;
                     | LDR X3, [X0, X5]      ;

exists(1:X2=(active:0) /\ 1:X3=0)

kinds: MP-fDI+W-dsb-W+R-addrpt-R Forbid
```

#### B1.22.3.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR X3,[X0]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` generates a DSB ST Effect $E_2$.
  - `GIC CDDI, X1` generates an Implicit Interrupt Read Effect $E_3$ to `INTID(A)` and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$ which resets the active state to 0.
- On P1:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.
  - `LDR X3,[X0,X5]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- There is an address dependency from $E_5$ to $E_7$ and, as a result, $E_5$ is Dependency-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before $E_4$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

### B1.22.4 Litmus test with control dependency

```
AArch64 MP-fDI+W-dsb-W+R-ctrlisb-R
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A); 1:X5=(active:1);
}
 P0                  | P1                    ;
 MOV     X3, #1      | GIC  CDRCFG, X1       ;
 STR     X3, [X0]    | ISB                   ;
 DSB     ST          | MRS  X2, ICC_ICSR_EL1 ;
 GIC     CDDI, X1    | TST  X2, X5           ;
                     | B.NE L0               ;
                     | ISB                   ;
                     | LDR  X3, [X0]        ;
                     | L0:                  ;
                     | NOP                   ;
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

```
exists(1:X2=(active:0) /\ 1:X3=0)

kinds: MP-fDI+W-dsb-W+R-ctrlisb-R Forbid
```

#### B1.22.4.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR X3,[X0]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` generates a DSB ST Effect $E_2$.
  - `GIC CDDI, X1` generates an Implict Interrupt Read Effect $E_3$ to `INTID(A)` and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$ which resets the active state to 0.
- On P1:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` E
  - `ISB` (the second instance) generates an Instruction Fetch Barrier E
  - `LDR X3,[X0]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- There is a control dependency from $E_5$ to and because $E_6$ is an Instruction Fetch Barrier Effect, $E_5$ is IFB-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

### B1.22.5 Litmus test without explicit synchronization

```
AArch64 MP-fDI+RR
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A);
}
 P0                  | P1                    ;
 MOV     X3, #1      | GIC CDRCFG, X1        ;
 STR     X3, [X0]    | ISB                   ;
 GIC     CDDI, X1    | MRS X2, ICC_ICSR_EL1  ;
                     | LDR X3, [X0]          ;

exists(1:X2=(active:0) /\ 1:X3=0)

kinds: MP-fDI+RR Allow
```

#### B1.22.5.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR X3,[X0]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `GIC CDDI, X1` generates an Implict Interrupt Read Effect $E_3$ to `INTID(A)` and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$ which resets the active state to 0.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- On P1:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.
  - `LDR X3,[X0]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.

This means that there is no cycle in the Ordered-before relation and, as a result, this execution satisfies the External visibility requirement. This execution is architecturally allowed.

### B1.22.6 Litmus test with a DSB but without a GSB

```
AArch64 MP-fDI+W-dsb-W+RR
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A);
}
 P0                  | P1                  ;
 MOV     X3, #1      | GIC CDRCFG, X1      ;
 STR     X3, [X0]    | ISB                 ;
 DSB     ST          | MRS X2, ICC_ICSR_EL1;
 GIC     CDDI, X1    |                     ;
                     | LDR X3, [X0]        ;

exists(1:X2=(active:0) /\ 1:X3=0)

kinds: MP-fDI+W-dsb-W+RR Allow
```

#### B1.22.6.1 Explanatio

In the execution that satisfies the post ndition:

- On P0:
  - `STR X` `[X0]` ge rates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `DSB ST` ge tes a DSB ST Effect $E_2$.
  - `GIC CDDI, X1` g tes an Implict Interrupt Read Effect $E_3$ to `INTID(A)` and an Implicit Interrupt Write Effe to `INTID(A)` $E_4$ which resets the active state to 0.
- C
  - G CDRCFG, X1 generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.
  - `LDR X3,[X0]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.
- $E_1$ is DSB-ordered-before $E_4$.

This means that there is no cycle in the Ordered-before relation and, as a result, this execution satisfies the External visibility requirement. This execution is architecturally allowed.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### B1.22.7 Litmus test without a DSB but with a GSB

```
AArch64 MP-fDI+WW+R-gsb-R
{
 [INTID(A)]=(active:1);
 0:X0=x; 0:X1=(intid:A);
 1:X0=x; 1:X1=(intid:A);
}
 P0                   | P1                   ;
 MOV     X3, #1       | GIC CDRCFG, X1       ;
 STR     X3, [X0]     | ISB                  ;
 GIC     CDDI, X1     | MRS X2, ICC_ICSR_EL1 ;
                      | GSB SYS              ;
                      | LDR X3, [X0]         ;

exists(1:X2=(active:0) /\ 1:X3=0)

kinds: MP-fDI+WW+R-gsb-R Allow
```

#### B1.22.7.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `STR X3,[X0]` generates an Explicit Memory Write Effect to `INTID(A)` $E_1$.
  - `GIC CDDI, X1` generates an Implicit Interrupt Read Effect to `INTID(A)` and an Implicit Interrupt Write Effect to `INTID(A)` $E_4$ which resets the active state to 0.
- On P1:
  - `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.
  - `GSB SYS` generates a GSB SYS Effect $E_6$.
  - `LDR X3,[X0]` generates an Explicit Memory Read Effect $E_7$.

In this execution:

- $E_5$ Reads-from $E_4$ and, as a result, $E_4$ is GIC-observed-by $E_5$.
- $E_5$ is GSB-ordered-before $E_7$.
- $E_7$ is Coherence-before $E_1$ and, as a result, $E_7$ is Explicit-observed-by $E_1$.

This means that there is no cycle in the Ordered-before relation and, as a result, this execution satisfies the External visibility requirement. This execution is architecturally allowed.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.23 IPI edge merging and message passing

## B1.23.1 Notes

The purpose of this test is to show that it is possible to inform a different PE that a GIC operation has completed by using a `GSB SYS` and a flag.

## B1.23.2 Litmus test

```
AArch64 edge-merging.ipi.MP
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P1);
 0:X0=(intid:A, pending:1);
 0:X1=x;
 1:X1=x;
 1:X3=(intid:A);
}
 P0                     | P1                        ;
 GIC     CDPEND, X0     | LDR  W0, [X1]             ;
 GSB     SYS            | DSB  LD                   ;
 MOV     X2, #1         | GICR X2, CDIA             ;
 STR     W2, [X1]       | GIC  CDRCFG, X3           ;
                        | ISB                       ;
                        | MRS  X4, ICC_ICSR_EL1     ;

exists(1:X0=1 /\ 1:X2=(valid:1, intid:A) /\
       1:X4=(pending:1, active:1, affinity:P1))

kinds: edge-merging.ipi.MP Forbid
```

### B1.23.2.1 Explanation

In the execution that satisfies the postcondition:

- On P0:
  - `GIC CDPEND,X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets the pending state to 1.
  - `GSB SYS` generates a GSB ACK Effect $E_3$.
  - `STR W2,[X1]` generates an Explicit Memory Write Effect $E_4$.
- On P1:
  - `LDR W0,[X1]` generates an Explicit Memory Read Effect to `INTID(A)` $E_5$.
  - `DSB LD` generates a DSB LD Effect $E_6$.
  - `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_7$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_8$ that resets the pending state to 0 and sets the active state to 1.
  - `GIC CDRCFG,X1` generates an Explicit Interrupt Read Effect $E_9$ to `INTID(A)`.

In the execution that satisfies the postcondition:

- $E_9$ Reads-from $E_2$ and, as a result, $E_7$ is Coherence-before $E_2$ and $E_7$ is GIC-observed-by $E_2$.
- $E_2$ GSB-ordered-before $E_4$
- $E_5$ Reads-from-memory $E_4$ and, as a result, $E_4$ is Explicit-observed-by $E_5$.
- $E_5$ is DSB-ordered-before $E_7$.

This creates a cycle in the Ordered-before relation and, as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.24 Device edge merging with GSB ACK

## B1.24.1 Notes

The purpose of this test is to show that two separate instances of an LPI being Pending are not allowed to be merged across an acknowledgement.

The LPI is initially Pending, it is then successfully acknowledged by `GICR X1, CDIA` which writes Pending = 0.

The STR has an implicit effect of writing Pending = 1, which happens in finite time, and the GIC will forward the Pending interrupt in finite time so that the second `GICR X2, CDIA` will return valid == 1 if not merged with the first instance.

Note that this test does not depend on the interrupt becoming Pending or being forwarded to the PE as a direct result of the write to the peripheral. The test relies solely on the finite tim[illegible]ntees which are covered by the `WAIT` construct.

See also:

- B1.1 *Interrupt litmus test assumptions*

## B1.24.2 Litmus test

```
AArch64 edge-merging.LPI+gsb.ack
{
 [INTID(A)]=(enabled:1,pending:1,affinity:0);
 [PTE(x)]=(oa:PA(x),attrs:(device-nGnRn[illegible]; // PE[illegible]P
 0:X0=x;
}
 P0
      GICR  X1, CDIA              ;
      GSB   ACK                   ;
      MOV   X3, #1                ;
      STR   X3, [X0]              ; trigger intid A via device
      GIC   CDDI, X1              ;
 WAIT(GICR  X2, CDIA  X2=(valid:1))

exists(0:X1=(val[illegible]:1) /\ [illegible]X2=(valid:0))

kinds: edge-merg[illegible]I[illegible]+gsb.ack [illegible]orbid
```

### B1.24.2.1 Explanation

In t[illegible]execut[illegible]atisfies the postcondition, on P0:

- `GIC[illegible]X1, CDIA` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_2$ which resets the pending state to 0 and sets the active state to 1.
- `GSB ACK` generates a GSB ACK Effect $E_3$.
- `STR X3, [X0]` generates an Explicit Write Effect $E_4$ to Location `x`. When the Explicit Write Effect to Location `x` reaches its endpoint, the Peripheral device signals an interrupt event to the IRI which generates Interrupt Read Effect $E_4$ to `INTID(A)` and an Interrupt Write Effect to `INTID(A)` $E_5$ which sets the pending state to 1.
- `GIC CDDI, X1` generates Implicit Interrupt Write Effect to `INTID(A)` $E_6$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_7$ which resets the pending state to 0.
- The repeated execution of `GICR X2,CDIA` generates Implicit Interrupt Read Effects $E_8, E_9, \ldots, E_n$ where n is a finite number.

All Interupt Write Effects are required to complete in finite time and consequently, Interrupt Write Effects $E_2$, $E_5$ and $E_7$ are Coherence-before $E_n$. In the execution that satisfies the postcondition, no interrupt is acknowledged and therefore `INTID(A)` does not become the HPPI. As a result one of the following has to be true:

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- $E_5$ is Coherence-before $E_2$ and consequently, $E_1$ Reads-from E~5 and $E_5$ is GIC-observed-by $E_1$. $E_1$ is GSB-ordered-before $E_5$. As a result, this execution violates the external visibility requirement as it creates a cycle in the Ordered-before relation.
- $E_7$ is Coherence-before $E_2$ and consequently, $E_1$ Reads-from E~7 and $E_7$ is GIC-observed-by $E_1$. $E_1$ is GSB-ordered-before $E_7$. As a result, this execution violates the external visibility requirement as it creates a cycle in the Ordered-before relation.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# B1.25 Configuration read and interrupt acknowledge

## B1.25.1 Notes

The purpose of these tests is to show that if an interrupt is disabled, and that disable is observed by a configuration read, then the interrupt cannot be acknowledged after observing the disable.

We show both a single-threaded and multi-threaded version of the test.

Illustrates a common principle of an interrupt acknowledge observing side effects of a write if that write can be observed via an explicit read.

## B1.25.2 Single-threaded litmus test

```
AArch64 coRR.ack
{
 [INTID(A)]=(enabled:1,affinity:P0,pending:1);
 0:X1=(intid:A);
}
 P0                    ;
 GIC  CDDIS, X1        ;
 GIC  CDRCFG, X1       ;
 ISB                   ;
 MRS  X2, ICC_ICSR_EL1 ;
 GICR X0, CDIA         ;

exists(0:X0=(valid:1, intid:A))

kinds: coRR.ack Forbid
```

### B1.25.2.1 Explanation

On P0:

- `GIC CDDIS,X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ which sets the enabled state to 0
- `GIC CDRCFG,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$.
- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition:

- Reads-from the initial value of `INTID(A)` and, as a result, due to the atomicity requirement for successful pairs of Read Interrupt and Write Interrupt Effects, $E_5$ is Coherence-before $E_2$.
- Any one of the following applies:
  - $E_3$ Reads-from the initial state of `INTID` and, as a result, $E_3$ is Coherence-before $E_2$. This violates the requirements of the Coherence-before relation. As a result, this execution is architecturally forbidden.
  - $E_3$ Reads-from $E_5$. This violates the Reads-from relation. As a result, this execution is architecturally forbidden.

## B1.25.3 Multi-threaded litmus test

```
AArch64 coRR.ack+dis+Rack
{
 [INTID(A)]=(enabled:1,affinity:P1,pending:1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0                | P1                    ;
 GIC CDDIS, X1     | GIC  CDRCFG, X1       ;
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

```
                    | ISB                    ;
                    | MRS  X2, ICC_ICSR_EL1 ;
                    | GICR X0, CDIA         ;

exists(1:X2=(enabled:0, affinity:P1, pending:1) /\
       1:X0=(valid:1, intid:A))

kinds: coRR.ack+dis+Rack Forbid
```

### B1.25.3.1 Explanation

- `GIC CDDIS,X1` generates an Explicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ which sets the enabled state to 0

On P1:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INT (A)` $E_3$
- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `ID(A)` $E_4$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition:

- $E_3$ is in program order before $E_4$.
- $E_3$ and $E_4$ are to the same Interrupt Location `INTID(A)`.
- $E_4$ Reads-from the initial state of `INTID(` and as result E herence-before E~2. This means that $E_3$ is GIC-Hazard-Ordered-before $E_2$
- $E_3$ Reads-from $E_2$ and as a result, $E_2$ is G served-by $E_3$.

This creates a cycle in the Ordered efore lation d as a result, the execution violates the External visibility requirement.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.26 Atomicity of interrupt acknowledge

## B1.26.1 Notes

The purpose of this test is to show that the two interrupt write effects of acknowledging an interrupt are atomic with respect to any read effect to the interrupt.

## B1.26.2 Litmus test

```
AArch64 atomic-gic.ack
{
 [INTID(A)]=(enabled:1,affinity:P0,active:0,pending:1,handling_mode:edge);
 1:X0=(intid:A)
}
 P0              | P1                    ;
 GICR X0, CDIA   | GIC  CDRCFG, X0       ;
                 | ISB                   ;
                 | MRS  X1, ICC_ICSR_EL1 ;

exists(1:X1=(pending:1, active:1) \/
       1:X1=(pending:0, active:0))

kinds: atomic-gic.ack Forbid
```

### B1.26.2.1 Explanation

On P0:

- `GICR X2,CDIA` generates an Implicit Interrupt Write Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Read Effect to `INTID(A)` $E_2$ and atomically sets the `pending` state to 0 and the `active` state to 1.

On P1:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$.

In the execution that satisfies the postcondition, $E_3$ reads that either the `pending` state is 1 and the `active` state is 1, or the `pending` state is 0 and the `active` state is 0. Both postconditions violate the atomicity requirement of $E_2$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.27 Atomicity of interrupt disable and acknowledge

## B1.27.1 Notes

This test demonstrates the interaction between disabling (clearing Enabled) and acknowledging an INTID from different PEs.

Illustrates a common principle of interrupt acknowledgement observing side effects of a write if that write can be observed via an explicit read.

## B1.27.2 Litmus test

```
AArch64 atomic.ack.dis.gic+W+R
{
 [INTID(A)]=(enabled:1,affinity:P0,pending:1,active:0);
 1:X1=(intid:A); 2:X1=(intid:A);
}
 P0                 | P1                 | P2                 ;
 GICR X2, CDIA      | GIC CDDIS, X1      | GIC CDRCFG, X1     ;
                    |                    | ISB                ;
                    |                    | MRS X2, IC_ICSR_EL1;

exists(0:X2=(valid:1,intid:A) /\
       2:X2=(enabled:0, pending:1, active:0))

kinds: atomic.ack.dis.gic+W+R Forbid
```

### B1.27.2.1 Explanation

On P0:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTID(A)` $E_2$ and atomically sets the `pending` state to 0 and the `active` state to 1.

On P1:

- `GIC CDDIS,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ and atomically sets the `enabled` state to 0.

On P2:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition, $E_5$ Reads-from $E_4$ and $E_4$ is GIC-observed-by $E_5$. The postcondition also requires that $E_3$ Reads-from the initial state of `INTID(A)`. At the same time, $E_1$ Reads-from the initial state of `INTID(A)`. This violates the atomicity requirement for successful pairs of Read Interrupt and Write Interrupt Effects which should apply for the pairs of $E_3$, E~4 and $E_1$, $E_2$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.28 Interrupt handler completion

## B1.28.1 Notes

The purpose of this test is to show that when an interrupt can be acknowledged on a PE, then disabling the interrupt on a different PE followed by reading back the interrupt configuration and state, exactly one of the following is true:

- The interrupt is observed as Active and P0 has acknowledged the interrupt.
- The interrupt is not observed as Active and P0 will not acknowledge the interrupt.

This illustrates a concept that operating systems would rely on, which is that if they've disabled the interrupt and observe an interrupt is not active, then that interrupt cannot be taken and become active without the interrupt being enabled again.

## B1.28.2 Litmus test

```
AArch64 atomic.ack.gic-ib+W-R-si-R
{
 [INTID(A)]=(enabled:1,affinity:P0,pending:1,active:0
 1:X1=(intid:A);
}
 P0                  | P1                      ;
 GICR X2, CDIA       | GIC CDDIS, X1
                     | GIC CDRCFG, X1          ;
                     | ISB                     ;
                     | MRS X2, ICC_ICSR_      ;

exists(0:X2=(valid:1, intid:A) /\
       1:X2=(enabled:0, affinity:P0, p ding:1 ctive:0))

kinds: atomic.ack.gic-ib+W-R -R Forbid
```

### B1.28.2.1 Explanation

On P0:

- `GICR X2,C A` gener s an Implicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Write Effect to `INTI A` $E_2$ and atomically sets the `pending` state to 0 and the `active` state to 1.

On P1:

- `GIC CDDI ,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write ffect `INTI` $E_4$ and atomically sets the `enabled` state to 0.
- `GIC RCFG, X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the executi that satisfies the postcondition:

- $E_1$ Reads-from the initial value of `INTID(A)` and, as a result, $E_1$ is Coherence-before $E_4$.
- E-5~ Reads-from $E_3$.
- $E_5$ is Coherence-before $E_2$ and, as a result, $E_4$ is also Coherence-before E~2.

This execution violates the atomicity requirement for the successful pair of Read Interrupt Effect $E_1$ and Write Interrupt Effect $E_2$, as $E_1$ is Coherence-before $E_4$ and $E_4$ is Coherence-before $E_2$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.29 Configuration update while disabled

## B1.29.1 Notes

The purpose of this test is to show that if multiple writes occur to the same interrupt while the interrupt is disabled, the interrupt will not be acknowledged and observe a partially updated interrupt.

## B1.29.2 Litmus test

```
AArch64 coWWW-gic.dis-en+ack+ack
{
 [INTID(A)]=(enabled:1, affinity:P1, active:0, pending:1, priority:5);
 0:X0=(intid:A);
 0:X1=(intid:A, priority:4);
 0:X2=(intid:A, affinity:P2);

 1:X1=(intid:A);
}
 P0               | P1                    | P2                    ;
 GIC CDDIS, X0    | GICR X2, CDIA         | GICR X2, CD           ;
 GIC CDPRI, X1    | ISB                   | ISB                   ;
 GIC CDAFF, X2    | MRS  X0, ICC_HAPR_EL1 | MRS  X0, ICC_HAPR_EL1 ;
 GIC CDEN, X0     |                       |                       ;

exists((1:X2=(valid:1, intid:A) /\ 1:X0=(pri          )) \/
       (2:X2=(valid:1, intid:A) /\ 2:X0=(p    ority:  )))

kinds: coWWW-gic.dis-en+ack+ack Forbid
```

### B1.29.2.1 Explanation

On P0:

- `GIC CDDIS,X0` genera s an Explicit Interrup Read Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ and sets `enabled` state to 0.
- `GIC CDPRI,X1` enerates an Explicit terrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ d sets its `priotity` to 4.
- `GIC CDAFF,X2` rates an Explicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_6$ and sets its `affininity` to P2.
- `GIC CDEN,X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_7$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_8$ and sets its `enabled` state to 1.

On P1:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_9$.

On P2:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_{10}$.

In the execution that satisfies the postcondition, any of the following applies:

- $E_4$ is Coherence-before $E_2$, $E_2$ is Coherence-before $E_6$ and E~8. $E_9$ Reads-from $E_4$. In this execution, $E_3$ is Coherence-before $E_2$ which violates the requirements of the Coherence-before relation.
- $E_2$ is Coherence-before $E_4$, $E_4$ is Coherence-before $E_8$, $E_8$ is Coherence-before $E_6$. $E_9$ Reads-from $E_8$. In this execution, $E_7$ is Coherence-before $E_6$ which violates the requirements of the Coherence-before relation.
- $E_6$ is Coherence-before $E_2$, $E_2$ is Coherence-before $E_4$ and E~8. $E_{10}$ Reads-from $E_6$. In this execution, $E_5$ is Coherence-before $E_2$ which violates the requirements of the Coherence-before relation.
- $E_2$ is Coherence-before $E_6$, $E_6$ is Coherence-before $E_8$, $E_8$ is Coherence-before $E_4$. $E_{10}$ Reads-from $E_8$. In this execution, $E_7$ is Coherence-before $E_4$ which violates the requirements of the Coherence-before relation.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.30 Atomicity of interrupt acknowledge and retarget

## B1.30.1 Notes

The purpose of this test is to show that even though an interrupt can be acknowledged and subsequently made pending, it cannot be acknowledged a second time without a deactivate of the interrupt taking place.

## B1.30.2 Litmus test

```
AArch64 co.ack-gic+WP+ack+ack
{
 [INTID(A)]=(enabled:1,affinity:P1,pending:1,active:0);
 0:X0=(intid:A, pending:1);
 0:X1=(intid:A, affinity:P2);
}
 P0               | P1               | P2              ;
 GIC CDAFF, X1    | GICR  X2, CDIA   | GICR X2, CDIA   ;
 GIC CDPEND, X0   |                  |                 ;

exists(1:X2=(valid:1,intid:A) /\ 2:X2=(valid:1,intid:A)

kinds: co.ack-gic+WP+ack+ack Forbid
```

### B1.30.2.1 Explanation

On P0:

- `GIC CDAFF,X1` generates an Explicit Interrupt [illegible] `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets its affinity to P2.
- `GIC CDPEND,X0` generates an Explicit Interrupt [illegible]ad Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ that sets its `pending` state to 1.

On P1:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$ and an Implicit Interrupt Read Effect to `INTID(A)` $E_6$ thats resets its `pending` state to 0 and its `active` state to 1.

On P3:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_7$ and an Implicit Interrupt Read Effect to `INTID(A)` that resets its `pending` state to 0 and its `active` state to 1.

In one of the possible executions that satisfies the postcondition:

- $E_5$ reads-from the initial state of `INTID(A)` and as result $E_5$ is Coherence-before $E_2$.
- $E_7$ Reads from $E_2$.
- $E_2$ is Coherence-before $E_6$.

This execution violates the atomicity requirement for the successful pair of Read Interrupt Effect $E_5$ and Write Interrupt Effect $E_6$, as $E_5$ is Coherence-before $E_2$ and $E_2$ is Coherence-before $E_6$.

There are other executions that satisfy the postcondition all of which violate the atomicity requirement for a successful pair of Read Interrupt and Write Interrupt Effects.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.31 1ofN interrupt acknowledge

## B1.31.1 Notes

The purpose of this test is to show that a 1ofN interrupt can only be acknowledged on a single PE.

## B1.31.2 Litmus test

```
AArch64 co.ack.1ofN-gic
{
 [INTID(A)]=(enabled:1,affinity:1ofN,pending:1,active:0);
}
 P0                | P1                  ;
 GICR X2, CDIA     | GICR  X2, CDIA      ;

exists(0:X2=(valid:1,intid:A) /\ 1:X2=(valid:1,intid:A))

kinds: co.ack.1ofN-gic Forbid
```

### B1.31.2.1 Explanation

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Implicit Interrupt Read Effect to `INTID(A)` $E_2$ thats resets its `pending` state to 0 and its `active` state to 1.

On P3:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Implicit Interrupt Read Effect to `INTID(A)` $E_4$ thats resets its `pending` state to 0 and its `active` state to 1.

In one of the execution that satisfies the postcondition:

- $E_1$ Reads-from the initial state of `INTID(A)` and as result $E_1$ and $E_2$ are Coherence-before $E_4$.
- $E_3$ Reads-from the initial state of `INTID(A)` and as result $E_3$ and $E_4$ are Coherence-before $E_2$.

This execution violates the atomicity requirement for the successful pair of Read Interrupt Effect $E_1$ and Write Interrupt Effect $E_2$, as $E_1$ is Coherence-before $E_4$ and $E_4$ is Coherence-before $E_2$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.32 Retargeting interrupts without synchronization

## B1.32.1 Notes

The purpose of this test is to show that due to the total order on writes to the same interrupt location, a write to change an interrupt's Affinity followed by a write to make the interrupt Pending, results in an interrupt not being taken on the old PE.

## B1.32.2 Litmus test

```
AArch64 coWW.ack-gic+WP+ack
{
 [INTID(A)]=(enabled:1,affinity:P1,pending:0,active:0);
 0:X0=(intid:A, pending:1); 0:X1=(intid:A, affinity:P0);
 1:X0=x; 1:X1=(intid:A);
}
P0                | P1               ;
GIC CDAFF, X1     | GICR  X2, CDIA   ;
GIC CDPEND, X0    |                  ;

exists(1:X2=(valid:1,intid:A))

kinds: coWW.ack-gic+WP+ack Forbid
```

### B1.32.2.1 Explanation

On P0:

- `GIC CDAFF,X1` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ that sets its affinity to P0.
- `GIC CDPEND,X0` generates an Explicit Interrupt Read Effect to `INTID(A)` $E_3$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_4$ that sets its `pending` state to 0.

On P1:

- `GICR X2,CDIA` generates an Implicit Interrupt Read Effect to `INTID(A)` $E_5$.

In the execution that satisfies the postcondition:

- $E_5$ Reads from $E_1$
- $E_4$ is Coherence-before $E_2$ and, as a result, $E_3$ is also Coherence-before E~2. This violates the requirements of the Coherence-before relation and, as a result, this is an architecturally forbidden execution.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.33 Reading interrupt configuration and exception status

## B1.33.1 Notes

The purpose of these tests is to show the interaction between observing a change to an interrupt configuration and observing a corresponding change in the interrupt exception pending status.

See also:

- B1.6 *Configuration and acknowledgement*

## B1.33.2 Litmus test with ISB before reading IRQ pending status

```
AArch64 coRex-gic+dis+RisbR
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0                    | P1                    ;
 GIC  CDDIS, X1        | GIC  CDRCFG, X1       ;
                       | ISB                   ;
                       | MRS  X2, ICC_ICSR_EL1 ;
                       | MRS  X0, ISR_EL1      ;

exists(1:X2=(enabled:0) /\ 1:X0=(I:1))

kinds: coRex-gic+dis+RisbR Forbid
```

### B1.33.2.1 Explanation

For the execution that validates the postcondition:

On P0:

- `GIC CDDIS,X1` generates an Explicit Interrupt Write Effect $E_1$ to `INTID(A)`.

On P1:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect $E_2$ to `INTID(A)`.
- `ISB` generates an Instruction Fetch Barrier Effect $E_3$.
- `MRS X0, ISR_EL1` generates a Direct System Register Read Effect to ISR_EL1 $E_4$.

In this execution, $E_2$ fulfils the postcondition, $E_2$ Reads-from $E_1$. But $E_4$ indicates that `INTID(A)` is still the HPPI. However, because of $E_3$ `INTID(A)` is disabled and cannot be the HPPI. As a result, this execution is architecturally forbidden.

## B1.33.3 Litmus test with ISB after reading IRQ pending status

```
AArch64 coRex-gic+dis+RR
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0                    | P1                    ;
 GIC  CDDIS, X1        | GIC  CDRCFG, X1       ;
                       | MRS  X0, ISR_EL1      ;
                       | ISB                   ;
                       | MRS  X2, ICC_ICSR_EL1 ;

exists(1:X2=(enabled:0) /\ 1:X0=(I:1))

kinds: coRex-gic+dis+RR Allow
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### B1.33.3.1 Explanation

For the execution that validates the postcondition:

On P0:

- `GIC CDDIS,X1` generates an Explicit Interrupt Write Effect $E_1$ to `INTID(A)`.

On P1:

- `GIC CDRCFG, X1` generates an Explicit Interrupt Read Effect $E_2$ to `INTID(A)`.
- `MRS X0, ISR_EL1` generates a Direct System Register Read Effect to ISR_EL1 $E_4$.

In the execution that satisfies the postcondition, $E_2$ Reads-from $E_1$. But $E_4$ indicates that `INTID(A)` is still the HPPI. This execution is architecturally allowed.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# B1.34 Reading interrupt configuration and IRQ unmask in PSTATE

## B1.34.1 Notes

The purpose of these tests is to show the interaction between observing a change to an interrupt configuration and observing the corresponding interrupt exception.

See also:

- B1.9 *Configuration write and IRQ unmask in PSTATE*

## B1.34.2 Explanation

The interrupt read effect R1 of `GIC CDRCFG, X1` on P1 may or may not observe the interrupt write effect W2 of `GIC CDDIS, X1` on P0, but if it does, it means that the interrupt has been r[illegible] from P1 on completion of W2.

If R1 from `GIC CDRCFG, X1` observes W2 from `GIC CDDIS, X1`, th[illegible] effects of W[illegible] on the execution context (whether an IRQ exception is pending) must be observed by the instru[illegible]ons followin[illegible] the ISB. This means that when `DAIFClr PSTATE.I` is executed, the IRQ interrupt excepti[illegible] will no[illegible] taken.

## B1.34.3 Litmus test with ISB before unmasking IRQ in PSTATE

```
AArch64 coRR+gic.cddis+gic.cdrcfg-isb-clr.i
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0                    | P1                    ;
 GIC  CDDIS, X1        | GIC  CDRCFG, X1       ;
                       | ISB                   ;
                       | MRS  X2, ICC_ICSR_EL1 ;
                       | DAIFClr  PSTATE.I    ;

exists(1:X2=(enabled:0) /\ async(P1, IRQ)

kinds: coRR+gic.cddis+gic.cdrcfg-isb-clr.i Forbid
```

## B1.34.4 Litmus test with ISB after unmasking IRQ in PSTATE

```
AArch64 coRR+gic.cddis+gic.cdrcfg-clr.i-isb
{
 [INTID(A)]=(enabled:1, pending:1, affinity:P1);
 0:X1=(intid:A);
 1:X1=(intid:A);
}
 P0                    | P1                    ;
 GIC  CDDIS, X1        | GIC  CDRCFG, X1       ;
                       | DAIFClr  PSTATE.I     ;
                       | ISB                   ;
                       | MRS  X2, ICC_ICSR_EL1 ;

exists(1:X2=(enabled:0) /\ async(P1, IRQ)

kinds: coRR+gic.cddis+gic.cdrcfg-clr.i-isb Allow
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.35 PPI activate and System register read

## B1.35.1 Notes

The PPI active state is stored in System registers.

These tests show that an ISB is required to guarantee the updates to the System registers from acknowledge interrupts are visible to a System register read.

Illustrates a common principle that any instruction with effects to state held in System registers, including GICR and GIC instructions affecting PPI state, can be observed following an ISB.

## B1.35.2 Litmus test without ISB

```
AArch64 coWmrs.ack
{
 0:[ppi(0)]=(enabled:1,pending:1,active:0);
}
P0
GICR  X0, CDIA                 ;
MRS   X1, ICC_PPI_SACTIVER0_EL1 ;

exists(0:X0=(valid:1, initd:ppi0) /\ 0:X1=0)

kinds: coWmrs.ack Allow
```

### B1.35.2.1 Explanation

On P0:

- `GICR X0,CDIA` acknowleges `ppi(0)` and as a result generates an Indirect System Register Write Effect to ICC_PPI_SACTIVER0_EL1 $E_1$.
- `MRS X1, ICC_PPI_SACTIVER0_EL1` generates a Direct System Register Read Effect to ICC_PPI_SACTIVER0_EL1 $E_2$.

As per the postconsumption, $E_2$ is Coherence-before $E_1$ which is architecturally allowed. See 'Table D24-1 Synchronization requirements' in the Architecture Reference Manual for A-profile [1] for more information.

## B1.35.3 Litmus test with GSB

```
AArch64 coWmrs.ack+gsb
{
 0:[ppi(0)]=(enabled:1,pending:1,active:0);
}
P0
GICR  X0, CDIA                 ;
GSB   ACK                      ;
MRS   X1, ICC_PPI_SACTIVER0_EL1 ;

exists(0:X0=(valid:1, initd:ppi0) /\ 0:X1=0)

kinds: coWmrs.ack+gsb Allow
```

### B1.35.3.1 Explanation

On P0:

- `GICR X0,CDIA` acknowleges `ppi(0)` and as a result generates an Indirect System Register Write Effect to ICC_PPI_SACTIVER0_EL1 $E_1$.
- `GSB ACK` generates a GSB ACK Effect $E_2$.
- `MRS X1, ICC_PPI_SACTIVER0_EL1` generates a Direct System Register Read Effect to ICC_PPI_SACTIVER0_EL1 $E_3$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

As per the postcondition, $E_3$ is Coherence-before $E_1$ which is architecturally allowed. See 'Table D24-1 Synchronization requirements' in the Architecture Reference Manual for A-profile [1] for more information.

### B1.35.4 Litmus test with ISB

```
AArch64 coWmrs.ack+isb
{
 0:[ppi(0)]=(enabled:1,pending:1,active:0);
}
P0
GICR  X0, CDIA                 ;
ISB                            ;
MRS   X1, ICC_PPI_SACTIVER0_EL1 ;

exists(0:X0=(valid:1, initd:ppi0) /\ 0:X1=0)

kinds: coWmrs.ack+isb Forbid
```

#### B1.35.4.1 Explanation

On P0:

- `GICR X0,CDIA` acknowleges `ppi(0)` and as a result generates an Indirect System Register Write Effect to ICC_PPI_SACTIVER0_EL1 $E_1$.
- `ISB` generates an Instruction Fetch Barrier Effect which is a Context Synchronization Event.
- `MRS X1, ICC_PPI_SACTIVER0_EL1` generates a Direct System Register Read Effect to ICC_PPI_SACTIVER0_EL1 $E_3$.

As per the postcondition, $E_3$ is Coherence-before $E_1$. This violates the requirements of the Coherence-before definition. As a result this execution is architecturally forbidden.

See also 'Table D24-1 Synchronization requirements' in the Architecture Reference Manual for A-profile [1] for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.36 PPI disable and acknowledge

## B1.36.1 Notes

The PPI configuration is stored in System registers.

The purpose of the test is to show that the effects of updates to the execution context are completed by the ISB, including which PPIs can be the HPPI.

Illustrates a common principle that any instruction with effects to state held in System registers, including GICR and GIC instructions affecting PPI state, can be observed following an ISB.

## B1.36.2 Litmus test with ISB

```
AArch64 comsrR.ack+isb
{
 0:[ppi(0)]=(enabled:1,pending:1,active:0);
 0:X1=(ppi0:1);
}
P0
MSR   ICC_PPI_CENABLER0_EL1, X1 ;
ISB                             ;
GICR  X0, CDIA                  ;

exists(0:X0=(valid:1, initd:ppi0))

kinds: comsrR.ack+isb Forbid
```

### B1.36.2.1 Explanation

On P0:

- `MSR ICC_PPI_CENABLER0_EL1, X1` generates a Direct System Register Write Effect to ICC_PPI_CENABLER0_EL1 and disables `ppi(0)`.
- `ISB` generates an Instruction Fetch Barrier Effect $E_2$ which is a Context Synchronization Event.
- `GICR X0,CDIA` acknowledges `ppi(0)` and as a result generates an Indirect System Register Read Effect to ICC_PPI_CENABLER0_EL1 $E_3$.

As per the conditions $E_3$ is Coherence-before $E_1$ as $E_3$ finds `ppi(0)` still enabled. This violates the requirements of the Coherence-before definition. As a result this execution is architecturally forbidden.

See also 'Table B2.4.1 Synchronization requirements' in the Architecture Reference Manual for A-profile [1] for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.37 PPI acknowledgement

## B1.37.1 Notes

The PPI active state is stored in System registers.

The purpose of the test is to show that a GICR instruction that acknowledges a PPI generates an Indirect System Register Write to ICC_PPI_SPEND<n>_EL1.

## B1.37.2 Litmus test

```
AArch64 coWR-ppi+ia-isb-mrs.sprendr1
{
 0:[ppi(64)]=(enabled:1,pending:1,active:0); // Edge-triggered IMPDEF PPI 64
 0:X1=1;
}
P0
GICR  X0, CDIA                  ;
ISB                             ;
MRS   ICC_PPI_SPENDR1_EL1, X2   ;

exists(0:X2=1)

kinds: coWR-ppi+ia-isb-mrs.sprendr1 Forbid
```

### B1.37.2.1 Explanation

In the execution that satisfies the postcondition on P0:

- `GICR X1, CDIA` acknowledges `ppi(64)` and generates an Indirect System Register Write to ICC_PPI_SPEND1_EL1 $E_1$ and sets `Pend0` to 0.
- `ISB` generates an Instruction Fetch Barrier effect $E_2$ which is a Context Synchronization Event.
- `MRS ICC_PPI_SPEND1_EL1, X2` generates a Direct System Register Read to ICC_PPI_SPEND1_EL1 $E_3$.

As per the postcondition, $E_3$ is Coherent before $E_1$. This violates the requirements of the Coherence-before definition. As a result this execution is architecturally forbidden. See 'Table D24-1 Synchronization requirements' in the Architecture Reference Manual for A-profile [1] for more information.

## B1.37.3 Litmus test

```
AArch64 edge-merging.ppi
{
 0:[ppi(64)]=(enabled:1,pending:1,active:0); // Edge-triggered IMPDEF PPI 64
 0:X1=1;
}
P0
GICR  X0, CDIA                  ;
MSR   ICC_PPI_SPENDR1_EL1, X1   ;
MRS   ICC_PPI_SPENDR1_EL1, X2   ;

exists(0:X2=0)

kinds: edge-merging.ppi Forbid
```

### B1.37.3.1 Explanation

In the execution that satisfies the postcondition, on P0:

- `GICR X1, CDIA` acknowledges `ppi(64)` and generates an Indirect System Register Write to ICC_PPI_SPEND1_EL1 $E_1$ and sets `Pend0` to 0.
- `MSR ICC_PPI_SPEND1_EL1, X1` generates a Direct System Register Write to ICC_PPI_SPEND1_EL1 $E_2$ and sets `Pend0` to `1`.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- `MRS ICC_PPI_SPEND1_EL1, X2` generates a Direct System Register Read to ICC_PPI_SPEND1_EL1 $E_3$.

As per the postcondition, $E_3$ Reads-from $E_1$. This is architecturally forbidden as $E_1$ and $E_2$ are ordered without the need of synchronization, and $E_3$ has to Read-from $E_2$ as both are Direct System Register Effects.

See also *Arm® Architecture Reference Manual, for A-profile architecture*[1] Table D24-1 Synchronization requirements' for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.38 Write after changing resident VM

## B1.38.1 Notes

The purpose of this test is to show that an ISB is required to change the resident VM on a PE so that interrupt write effects are made to the expected interrupts.

Illustrates a common principle that any instruction with effects to state held in System registers, including GICR and GIC instructions affecting PPI state, can be observed following an ISB.

## B1.38.2 Litmus test

```
AArch64 vmW-gic+R
{
 [VM0:INTID(A)]=(enabled:0);
 [VM1:INTID(A)]=(enabled:0);
 0:ICH_CONTEXTR_EL2=(VM1, VPE0, VALID);
 0:X0=(intid:A); 0:X1=(VM0, VPE0, VALID);
 1:ICH_CONTEXTR_EL2=(VM1, VPE1, VALID);
 1:X0=(intid:A);
}
 P0                        | P1                        ;
 MSR ICH_CONTEXTR_EL2, XZR | GIC VDRCFG, X0           ;
 MSR ICH_CONTEXTR_EL2, X1  | ISB                       ;
 ISB                       | MRS X2, ICC_ICS[illegible] ;
 GIC VDEN, X0              |                           ;

exists(1:X2=(enabled:1))

kinds: vmW-gic+R Forbid
```

### B1.38.2.1 Explanation

On P0:

- `MSR ICH_CONTEXTR_EL2, XZR` generates a Direct System Register Write Effect to ICH_CONTEXTR_EL2 $E_1$ and ensures that there is no resident VPE.
- `MSR ICH_CONTEXTR_EL2, X1` generates a Direct System Register Write Effect to ICH_CONTEXTR_EL2 $E_2$ and makes VPE 0 of VM 0 resident.
- `ISB` generates an Instruction Fetch Barrier Effect $E_3$ which is a Context Synchronization Event.
- `GIC VDEN, X0` generates an Indirect System Register Read to ICH_CONTEXTR_EL2 $E_4$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_5$ enabling `INTID(A)`.

On P1:

- `GIC VDRCFG, X0` generates an Indirect System Register Read to ICH_CONTEXTR_EL2 $E_6$ and an Explicit Interrupt Read Effect to `INTID(A)` $E_7$.

As per the postcondition, $E_7$ reads that `INTID(A)` is enabled which could only have happened if $E_7$ Reads-from $E_5$. This in turn means that $E_5$ was to `VM1:INTID(A)` and as a result, $E_4$ Reads-from the initial value of ICH_CONTEXTR_EL2 and $E_4$ is Coherence-before $E_2$. This violates the requirements of the Coherence-before definition and this execution is architecturally forbidden.

See ‘Table D24-1 Synchronization requirements’ in the Architecture Reference Manual for A-profile [1] for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.39 Write before changing resident VM

## B1.39.1 Notes

The purpose of this test is to show that an ISB is not needed to ensure that interrupt write effects are made to the resident VM when an instruction to change the resident VM occurs in program order after the instruction causing the interrupt write effect.

## B1.39.2 Litmus test

```
AArch64 Wvm-gic+R
{
 [VM0:INTID(A)]=(enabled:0);
 [VM1:INTID(A)]=(enabled:0);
 0:ICH_CONTEXTR_EL2=(VM0, VPE0);
 0:X0=(intid:A); 0:X1=(VM1, VPE0);
 1:ICH_CONTEXTR_EL2=(VM1, VPE0);
 1:X0=(intid:A);
}
P0                        | P1                          ;
GIC VDEN, X0              | GIC VDRCFG, X0
MSR ICH_CONTEXTR_EL2, XZR | ISB
MSR ICH_CONTEXTR_EL2, X1  | MRS X2, ICC_ICSR_EL1        ;

exists(1:X2=(enabled:1))

kinds: Wvm-gic+R Forbid
```

### B1.39.2.1 Explanation

On P0:

- `GIC VDEN, X0` generates an Indirect System Register Read to ICH_CONTEXTR_EL2 $E_1$ and an Explicit Interrupt Write Effect to `INTID(A)` $E_2$ enabling `INTID(A)`.
- `MSR ICH_CONTEXTR_EL2, XZR` generates a Direct System Register Write Effect to ICH_CONTEXTR_EL2 $E_3$ and ensures that there is no resident VPE.
- `MSR ICH_CONTEXTR_EL2, X1` generates a Direct System Register Write Effect to ICH_CONTEXTR_EL2 $E_4$ and makes VPE0 of VM 0 resident.

On P1:

- `GIC VDRCFG, X0` generates an Indirect System Register Read to ICH_CONTEXTR_EL2 $E_5$ and an Explicit Interrupt Read Effect to `INTID(A)` $E_6$.

As per the precondition, $E_6$ reads that `INTID(A)` is enabled which could only have happened if $E_6$ Reads-from $E_2$. This in turn means that $E_2$ was to `VM1:INTID(A)` and as a result, $E_1$ Reads-from $E_4$. This violates the requirements of the Reads-from definition and this execution is architecturally forbidden.

See ‘Table D24-1 Synchronization requirements’ in the Architecture Reference Manual for A-profile [1] for more information.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.40 Completion of GIC and GICR instructions in finite time

## B1.40.1 Notes

The purpose of this test is to show that writes to Interrupt Locations are required to become visible to observers in finite time.

## B1.40.2 Litmus test with a priority update observed by a configuration read

```
AArch64 gic.pri+gic.rcfg-wait
{
 [INTID(A)]=(priority:0);
 0:X1=(intid:A, priority:1);
 1:X0=(intid:A); 1:X2=(priority:1);
}
 P0            | P1                   ;
 GIC CDPRI, X1 |L0:                   ;
               | GIC CDRCFG, X0       ;
               | ISB                  ;
               | MRS X1, ICC_ICSR_EL1 ;
               | CMP X1, X2          ;
               | B.NE L0             ;
existsN(1:X1=(priority:1))

kinds: gic.pri+gic.rcfg-wait Require
```

### B1.40.2.1 Explanation

On P0:

- `GIC CDPRI,X1` generates an Explicit Interrupt Write Effect $E_1$ to `INTID(A)`.

On P1:

- `GIC CDRCFG, X0` generates Explicit Interrupt Read Effects $E_2$, $E_3$, ... $E_n$ to `INTID(A)`.

It is architecturally required that there is a finite number `n` for which $E_n$ Reads-from $E_1$.

## B1.40.3 Litmus test with an interrupt acknowledge observed by a configuration read

```
AArch64 gic.ia+gic.rcfg-wait
{
 [INTID(A)]=(enabled:1, pending:1, active:0, priority:1, affinity:P0);
 1:X0=(intid:A);
 1:X2=(enabled:1, pending:0, active:1, priority:1, affinity:P0);
}
 P0            | P1                   ;
 GICR X1,CDIA  |L0:                   ;
               | GIC CDRCFG, X0       ;
               | ISB                  ;
               | MRS X1, ICC_ICSR_EL1 ;
               | CMP X1, X2          ;
               | B.NE L0             ;
existsN(0:X1=(valid:1, intid:A) /\ 1:X1=(priority:1))

kinds: gic.ia+gic.rcfg-wait Require
```

### B1.40.3.1 Explanation

On P0:

- `GICR X1,CDIA` generates an Implicit Interrupt Write Effect $E_1$ to `INTID(A)`.

On P1:

- `GIC CDRCFG, X0` generates Explicit Interrupt Read Effects $E_2$, $E_3$, ... $E_n$ to `INTID(A)`.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

It is architecturally required that there is a finite number `n` for which $E_n$ Reads-from $E_1$.

### B1.40.4 Litmus test with an interrupt becoming pending and then acknowledged

```
AArch64 gic.pend1+gic.ia-wait
{
 [INTID(A)]=(enabled:1, pending:0, active:0, priority:1, affinity:P1);
 0:X0=(intid:A,pending:1);
 1:X2=(valid:1,intid:A);
}
 P0            | P1                  ;
 GIC CDPEND,X0 |L0:                  ;
               | GICR X1,CDIA        ;
               | CMP X1, X2          ;
               | B.NE L0             ;
existsN(1:X1=(valid:1,intid:A))

kinds: gic.pend1+gic.ia-wait Require
```

#### B1.40.4.1 Explanation

On P0:

- `GIC CDPEND,X0` generates an Explicit Interrupt Write Effect $E_1$ to `INTID(A)`.

On P1:

- `GICR X1,CDIA` generates Implicit Interrupt Read Effects $E_2$, $E_3$ … $E_n$ to `INTID(A)`.

It is architecturally required that there is a finite number `n` for which $E_n$ Reads-from $E_1$.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.41 Effects of disabling a PPI source on the PPI Pending state

## B1.41.1 Notes

The purpose of this test is to show that updates to a PPI Pending state can not be synchronized by any instruction.

## B1.41.2 Litmus test with disable of the timer state

```
AArch64 timer-disable-isb-ppi-read
{
 0:[ppi(27)]=(enabled:1,pending:1,active:0);
}
 P0
 MRS   X0, CNTV_CTL_EL0          ;
 MOV   X1, #0                    ;
 MSR   CNTV_CTL_EL0, X1          ;
 ISB                             ;
 MRS   X2, ICC_PPI_SPENDR0       ;

exists(0:X0=(ISTATUS=1,IMASK=0,ENABLE=1) /\ 0:X2=0x00020000)

kinds: timer-disable-isb-ppi-read Allow
```

### B1.41.2.1 Explanation

Updates from a PPI source to the PPI Pending state is an autonomous asynchronous event.

Even though the PPI Pending state can be observed by a System register read of the ICC_PPI_SPENDR0_EL1 System register, the update to the PPI Pending state is not an indirect System register write to ICC_PPI_SPENDR0_EL1 caused by the direct System register write to CNTV_CTL_EL0.

Instead, the write to CNTV_CTL_EL0 is observed by the autonomous asynchronous event that causes the EL1 Virtual Timer to evaluate its state and update the Pending state of the PPI.

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# B1.42 Priority mask updates and interrupt pending state and acknowledgement

## B1.42.1 Notes

The purpose of these tests is to show the relationship between a write to ICx_PCR_ELx and the value returned by a read of ICx_HPPIR_ELx and acknowledging an interrupt.

The following litmus tests are all examples of the behavior described by rule NWMPV which specifies that masking an interrupt as the result of a direct write to the priority mask registers are visible to instructions in program order after the direct write without any explicit synchronization.

When an HPPI is of sufficient priority to be signaled as a result of a write to the priority mask registers, the interrupt is signaled and presented as the HPPI in finite time.

## B1.42.2 Litmus test masking using the priority mask without synchronization

```
AArch64 pcr-mask-async
{
 [INTID(A)]=(enabled:1, pending:1, priority:5);
 0:X0=4;
}
 P0                    ;
 MSR ICC_PCR_EL1, X0   ;
 MRS X1, ICC_HPPIR_EL1 ;

exists(0:X1=(hppiv:1, intid:A))

kinds: pcr-mask-async Forbid
```

## B1.42.3 Litmus test unmasking using the priority mask without synchronization

```
AArch64 pcr-unmask-async
{
 [INTID(A)]=(enabled:1, pending:1, priority:5);
 0:X0=31;
 0:ICC_PCR_EL1=4;
}
 P0
 MSR ICC_PCR_EL1,      ;
 MRS X1,  PPIR_EL1     ;

exists (0:X1=(hppiv:0))

kinds: pcr-unmask-async Allow
```

## B1.42.4 Litmus test masking using the priority mask and acknowledging without synchronization

```
AArch64 pcr-mask-ack
{
 [INTID(A)]=(enabled:1, pending:1, priority:5);
 0:X0=4;
}
 P0                    ;
 MSR ICC_PCR_EL1, X0   ;
 GICR X1, CDIA         ;

exists(0:X1=(valid:1, intid:A))

kinds: pcr-mask-ack Forbid
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

### B1.42.5 Litmus test unmasking using the priority mask and acknowledging without synchronization

```
AArch64 pcr-mask-ack
{
 [INTID(A)]=(enabled:1, pending:1, priority:5);
 0:X0=31;
 0:ICC_PCR_EL1=4;
}
 P0                    ;
 MSR ICC_PCR_EL1, X0   ;
 GICR X1, CDIA         ;

exists(0:X1=(valid:0))

kinds: pcr-mask-ack Allow
```

RETIRED


*Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

RETIRED

# Part C
## Model

# Chapter C1
# Operational model

RETIRED

IQZJCC The following code is an operational model of the GIC, written in Arm Specification Language (ASL). This section is is at Alpha quality. Alpha quality means that most major features of the specification are described in this release, but some features and details might be missing.

See also:

- Arm Specification Language Reference Manual [13].

```
// exceptions.asl
//
// This is the pseudocode implementing the exceptions signaling logic.

INTID GetHPPI(bits(2) domain)
    return DOMAIN_HPPIS[UInt(domain)].intid;

boolean HasHPPI(bits(2) domain)
    return DOMAIN_HPPIS[UInt(domain)].valid;

boolean DomainEnabled(bits(2) domain)
    if domain == DOM_EL3 then
        return ICC_CR0_EL3.EN == '1';
    else
        return BankedICC_CR0_EL1(domain).EN == '1';

integer GetRunningPriority(bits(2) domain)
    return 31; // Should return the running priority based on the value in the banked ICC_APR_EL1
        ↪or ICC_APR_EL3

integer GetPriorityMask(bits(2) domain)
    case domain of
        when DOM_EL3 return UInt(ICC_PCR_EL3.Priority);
        when DOM_S return UInt(ICC_PCR_EL1[0].Priority);
```


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        when DOM_NS return UInt(ICC_PCR_EL1[1].Priority);
        when DOM_RL return UInt(ICC_PCR_EL1[2].Priority);

boolean HPPIAvailable(bits(2) domain)
    if !DomainEnabled(domain) then
        return FALSE;

    hppi = GetHPPI(domain);
    hppi_priority = UInt(hppi.Priority);
    running_priority = GetRunningPriority(domain);
    priority_mask = GetPriorityMask(domain);

    return hppi_priority < running_priority && hppi_priority <= priority_mask;

bits(2) PreemptiveDomain()
    return ICC_CR0_EL3.PID;

boolean PreemptiveHPPIAvailable(bits(2) domain)
    if domain == DOM_EL3 then
        return FALSE; // 0b10 in ICC_CR0_EL3.PID encodes no preemptive domain

    if PreemptiveDomain() != domain then
        return FALSE;

    return HPPIAvailable(domain) && UInt(GetHPPI(domain).Priority) <
        ↪UInt(BankedICC_CR0_EL1(domain).IPPT);


// GICSignalException()
// ====================
// Signal CPU interrupt exception based on HPPI availability in the Interrupt
// Domains.

GICPhysicalExceptions()
    if CurrentDomain() == DOM_EL3 then
        if HPPIAvailable(DOM_EL3) ||
           HPPIAvailable(DOM_S)
           HPPIAvailable(DOM_NS) ||
           HPPIAvailable(DOM_RL) then
            FIQ();
    else // Non-EL3 Domain
        if HPPIAvailable(EL3) || PreemptiveHPPIAvailable(PreemptiveDomain()) then
            FIQ();
        elsif HPPIAvailable(CurrentDomain()) then
            IRQ();

    return;

// gic_shared_functions.asl

sysreg_ICC_CR0_EL1 BankedICC_CR0_EL1(bits(2) domain)
    case domain of
        when DOM_EL3 assert FALSE;
        when DOM_S return ICC_CR0_EL1[0];
        when DOM_NS return ICC_CR0_EL1[1];
        when DOM_RL return ICC_CR0_EL1[2];

bits(2) CurrentDomain()
    if PSTATE.EL == EL3 then
        return DOM_EL3;
    else
        bits(2) state_bits;
        if HaveRME() then
            state_bits = SCR_EL3.<NSE,NS>;
        else
            state_bits = '0' : SCR_EL3.NS;

        case state_bits of
            when '00' return DOM_S;
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            when '01' return DOM_NS;
            when '10' Unreachable();
            when '11' return DOM_RL;


// HavePreemptiveDomain()
// ======================
// Returns TRUE when the Domain selected by ICC_CR0_EL3 is implemented, FALSE otherwise

// MostPrivilegedSecurityState()
// =============================
// Returns the most privileged Security state

SecurityState MostPrivilegedSecurityState()
    // We can always ask SecurityStateAtEL with 'EL3' to get the right answer even if EL3 is not
        ↪implemented
    return SecurityStateAtEL(EL3)

// IsAccessMPPAS()
// ===============
// Returns TRUE when an access is made with the Most Privileged PAS

boolean IsAccessMPPAS();
    mpss = MostPrivilegedSecurityState()

    case mpss of
        when SS_Root return IsAccessRoot();
        when SS_Secure return IsAccessSecure();
        when SS_NonSecure return IsAccessNonSecure();
        when SS_Realm Unreachable();


// IsAccessIWBMPPAS()
// ==================
// Returns TRUE when an access is made with the Most Privileged PAS of the IWB

boolean IsAccessIWBMPPAS();
    iwb_mppas = IWBMPPAS()

    case iwb_mppas of
        when PAS_Root return IsAccessRoot();
        when PAS_Secure return IsAccessSecure();
        when PAS_NonSecure return IsAccessNonSecure();
        otherwise Unreachable();


bits(2) IWBMPPAS()
    assert IWB_IDR.INT_DOMS
    case domains of
        when '0001' return PAS_Secure;
        when '0010' return PAS_NonSecure;
        when '0111' return PAS_Secure;
        when '1110' return PAS_Root;
        when '1111' return PAS_Root;
        otherwise Unreachable();


// IsWireAccessible()
// ==================
// Returns TRUE when a wire can be configured on an IWB for a given access PAS

boolean IsWireAccessible(bits(11) n, bits(5) x)
    // IWB MPPAS can access state of all wires
    if IsAccessIWBMPPAS() then
        return TRUE;

    reg_index = (n * 2) + (x / 16);
    wire_index = x MOD 16;
    wire_domain = IWB_WDOMAINR[reg_index].WDOM[wire_index];
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    if IsAccessRoot() then
        // If PAS is Root and it is not the IWB MPPAS, then either EL3 Domain is
        // not supported or it is associated with the Secure PAS.
        return FALSE;

    if IsAccessSecure() then
        // Wire cannot be accessed if Secure Interrupt Domain is not implemented.
        if IWB_IDR0.INT_DOMS[0] == 0 then
            return FALSE;
        if wire_domain == DOM_S then
                return TRUE;

    if IsAccessRealm() then
        // Wire cannot be accessed if Realm Interrupt Domain is not implemented.
        if IWB_IDR0.INT_DOMS[3] == 0 then
            return FALSE;
        if wire_domain == DOM_RL then
            return TRUE;

    if IsAccessNonSecure() then
        // Wire cannot be accessed if Non-secure Interrupt Domain is not implemented.
        if IWB_IDR0.INT_DOMS[1] == 0 then
            return FALSE;
        if wire_domain == DOM_NS then
            return TRUE;

    Unreachable();


// IsWireDomainRO()
// ================
// Returns TRUE if the Interrupt Domain assignment of an IWB input wire is fixed

boolean IsWireDomainRO(bits(16) wire_idx)
    return IMPLEMENTATION_DEFINED "Whether the Interrupt Domain assignment of a wire is fixed.";


// IsWireConfigRO()
// ================
// Returns TRUE if the Trigger mode of an IWB input wire is read-only

boolean IsWireConfigRO(bits(16) wire_idx)
    return IMPLEMENTATION_DEFINED "Whether an IWB input wire Trigger mode is read-only";

// IsPPIImplemented()
// ==================
// Returns TRUE when a PPI is implemented.

boolean IsPPIImplemented(bits(7) PPI_ID)
    return IMPLEMENTATION_DEFINED "Whether a PPI is implemented";

// IsPPIAssignedToCurrentDomain()
// ==============================
// Returns TRUE when a physical PPI is assigned to the Current Physical Interrupt Domain

boolean IsPPIAssignedToCurrentDomain(bits(7) PPI_ID)
    n = PPI_ID DIV 32;
    x = PPI_ID MOD 32;

    if !HaveEL(EL3) then
        return TRUE;

    return ICC_PPI_DOMAINR_EL3[n]<x + 1:x> == CurrentDomain();


// IsSPIAssignedToDomain()
// ================
// Returns TRUE if the SPI is assigned to the specified domain.
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
// This would be correctly modeled by accessing the internal SPI Domain assignment state that is
    ↪exposed via IRS_SPI_DOMAINR.

boolean IsSPIDVISupported(bits(29) spi_id, bits(2) domain);


// IsSPIDVISupported()
// ===================
// Returns TRUE if the SPI supports being assigned to a VM.
// Arm strongly recommends that an SPI that can be driven by a signal external to the IRS supports
    ↪assignment to a VM.

boolean IsSPIDVISupported(bits(29) spi_id)
    return IMPLEMENTATION_DEFINED "Whether the SPI can be assigned to a VM.";
```

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Glossary

**Abort**

Any situation where the memory system returns an error response to a memory access by the GIC. Examples of error responses could include GPF faults returned by an SMMU[4] or DECERR in AMBA AXI[6] systems.

**Accepted**

An event has been Accepted by a GICv5 system component when it has sent a reply to the sender of the event. See 3.2 *Communication between GIC system components* for more information.

**Application PE**

A PE used by the operating system or hypervisor to execute user application or kernel threads.

**ASL**

Arm Specification Language
Language used to express pseudocode implementations. The formal language definition can be found in the *Arm® Specification Language Reference Manual*[13].

**Candidate HPPI**

An interrupt identified among a subset of all interrupts as a candidate for the HPPI. See 2.7 *Interrupt Prioritization* for more information.

**Doorbell PPI**

A PPI used to signal to an Interrupt Domain that there is an interrupt pending for another Interrupt Domain. See 2.9.6 *Doorbell PPIs* for more information.

**Downstream**

An interrupt signal flows *downstream* from the interrupt source through the ITS, IRS, and is presented to PEs.

Where two components are involved, *upstream* is the component farthest away from the PE, and *downstream* is the component closest to the PE.

This also applies to the GICv5 Stream protocol where downstream packets flow from the IRS to the PE, and upstream packets flow from the PE to the IRS.

**GPC**

Granule Protection Check. The process of checking whether an access to a Location is permitted by the Granule Protection Table. This term is also used to refer to an MMU-attached or SMMU-attached Granular PAS filter that implements the Granule Protection Check. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information.

**HPPI**

Highest Priority Pending Interrupt. See 2.7 *Interrupt Prioritization* for more information.

**IAFFID**

Interrupt Affinity ID, a value that uniquely indentifies PEs connected to a GIC. See 2.2 *The GICv5 CPU interface* for more information.

**IPI**

Inter-processor Interrupt. See 2.5 *Inter-Processor Interrupts* for more information.

**IRI**

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Interrupt Routing Infrastructure. See Chapter 1 *Introduction* and Chapter 3 *GICv5 system architecture* for more information.

**IRS**

Interrupt Routing Service. See Chapter 4 *Interrupt routing service (IRS)* for more information.

**ITS**

Interrupt Translation Service. See Chapter 5 *Interrupt translation service (ITS)* for more information.

**IWB**

Interrupt Wire Bridge. See Chapter 6 *Interrupt Wire Bridge (IWB)* for more information.

**LPI**

Logical Peripheral Interrupt. See 2.4 *Interrupt types and identifiers* for more information.

**MPPAS**

Most Privileged PAS. See 3.1 *Interrupt Domains* for more information.

**MPSS**

Most Privileged Security State. See 3.1 *Interrupt Domains* for more information.

**NMI**

Non-maskable Interrupt. See 2.9 *The physical CPU interface* for more information.

**PAS**

Physical Address Space. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] for more information.

**PPI**

PE-Private Peripheral Interrupt. See 2.4 *Interrupt types and identifiers* for more information.

**Processed**

An interrupt event has been processed by an IRS when the interrupt effects of an interrupt event can be observed by PEs. See 4.5 *Interrupt synchronization requests* for more information.

**RAS**

Reliability, Availability, and Serviceability. See *Arm® Architecture Reference Manual, for A-profile architecture*[1] and *Arm® Reliability, Availability, and Serviceability (RAS) System Architecture for A-profile architecture*[14] for more information.

**Reachable**

An INTID is reachable from a PE if the PE can access the configuration and state of the interrupt. See 2.6 *GIC System instructions* and 4.6 *Interrupt configuration and state* for more information.

**Running priority**

The highest active priority on an Interrupt Domain. See 2.8 *Interrupt handling* for more information.

**SGI**

Software Generated Interrupt. A dedicated interrupt type for IPIs in GIC version 3 and version 4. See *Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4*[3] for more information.

**SPI**

Shared Peripheral Interrupt. See 2.4 *Interrupt types and identifiers* for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential

**Unreachable**

See Reachable.

**Upstream**

See Downstream

**VM**

Virtual machine.

**VMSA**

Virtual Memory System Architecture defined in *Arm® Architecture Reference Manual, for A-profile architecture*[1].

**VPE**

Virtual PE.

**VPE doorbell**

An LPI which is signaled when a non-resident VPE receives an interrupt. See 4.1 *VPE doorbells* for more information.

**Write-One-to-Clear (W1C)**

Writes of 0 to the bit are ignored. A write of 1 clears the bit to 0. See *Arm Architecture Reference Manual, for A-profile architecture*[1] for more information.

RETIRED


Copyright © 2022-2025 Arm Limited or its affiliates. All rights reserved.
Non-confidential