ARM® Reliability, Availability, and Serviceability (RAS) Specification

ARMv8, for the ARMv8-A architecture profile

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.

Document number: DDI 0587B

ARM Reliability, Availability, and Serviceability (RAS) Specification

Copyright © 2017 ARM Limited or its affiliates. All rights Reserved.

Release information

The following changes have been made to this document.

Date	Issue	Confidentiality	Change
March 2017	A	Non-Confidential	First issue
September 2017	B	Non-Confidential	EAC Release

Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the Arm’s

trademark usage guidelines
http://www.arm.com/company/policies/trademarks.

Copyright © 2017 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

In this document, where the term ARM is used to refer to the company it means “Arm or any of its subsidiaries as appropriate”.

Confidentiality Status

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

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Contents

Preface **7**
References 7
Feedback 7

1 Introduction **8**
1.1 Types of errors 9
1.2 Techniques for improving reliability, availability, and serviceability 9
1.2.1 Fault prevention and fault removal 10
1.2.2 Error handling and recovery 10
1.2.3 Fault handling 10
1.3 General taxonomy of errors 10
1.3.1 Error detection 10
1.3.2 Error propagation 11
1.3.3 Infected, poisoned, containable, and uncontainable 11
1.4 Fault handling 11

2 ARMv8-A Error Handling and Recovery Architecture **12**
2.1 Taxonomy of errors in the ARMv8-A architecture 12
2.1.1 Architectural error detection 12
2.1.2 Architectural error propagation 12
2.1.3 Architecturally infected, containable, and uncontainable 13
2.1.4 Architecturally consumed errors 13
2.1.5 Other errors 14
2.2 Generating error exceptions 14
2.2.1 Error correction and deferment 14
2.2.2 Error exceptions 14
2.3 Error record System register view 16
2.3.1 Fields in VSESR_EL2, VDFSR, DISR(_EL1), and VDISR(_EL2) 17
2.4 Error Synchronization Barrier 18
2.4.1 ESB and Unrecoverable errors 18
2.4.2 ESB and other containable errors 19
2.4.3 ESB and other physical errors 19
2.4.4 ESB and virtual errors 19
2.4.5 Extension for barrier at exception entry and exit 20
2.5 Taking external abort and SError interrupt exceptions to AArch64 21
2.5.1 Target Exception level 21
2.5.2 AArch64 syndrome registers on taking an error exception 21
2.5.3 Multiple SError interrupts 21
2.6 Taking external abort and SError interrupt exceptions to AArch32 22
2.6.1 Target mode 22
2.6.2 AArch32 syndrome registers on taking an error exception 22
2.6.3 Multiple SError interrupts 22
2.7 Virtual SError interrupts 23

3 RAS System Architecture **24**
3.1 Recording errors 24

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

3.1.1 Detected uncorrected errors in data 24
3.1.2 Corrected errors in data 24
3.1.3 Other sources of error and warnings 25
3.1.4 Error types 25
3.1.5 Software faults 26
3.2 Standard error record 26
3.2.1 Nodes 27
3.2.2 Multiple records per node 28
3.2.3 Writing the error record 28
3.2.4 Security and Virtualization 31
3.2.5 Synchronization and error record accesses 32
3.3 Error recovery interrupt 32
3.4 Fault handling interrupt 32
3.5 In-band error signaling (external aborts) 33
3.6 Standard format Corrected error counter 33
3.7 Error recovery and fault handling signaling 34
3.8 Error Recovery reset 34

4 RAS Extension Registers 36
4.1 Memory-mapped view 36
4.2 Reset values 36
4.3 Writes to ERR<*n*>STATUS 37
4.4 AArch64 System registers 38
4.4.1 Register index 38
4.4.2 DISR_EL1, Deferred Interrupt Status Register 39
4.4.3 ERRIDR_EL1, Error Record ID Register 41
4.4.4 ERRSELR_EL1, Error Record Select Register 42
4.4.5 ERXADDR_EL1, Selected Error Record Address Register 44
4.4.6 ERXCTLR_EL1, Selected Error Record Control Register 45
4.4.7 ERXFR_EL1, Selected Error Record Feature Register 46
4.4.8 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0 47
4.4.9 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1 48
4.4.10 ERXSTATUS_EL1, Selected Error Record Primary Status Register 49
4.4.11 VDISR_EL2, Virtual Deferred Interrupt Status Register 50
4.4.12 VSESR_EL2, Virtual SError Exception Syndrome Register 53
4.5 AArch32 System registers 54
4.5.1 Register index 54
4.5.2 DISR, Deferred Interrupt Status Register 55
4.5.3 ERRIDR, Error Record ID Register 58
4.5.4 ERRSELR, Error Record Select Register 59
4.5.5 ERXADDR, Selected Error Record Address Register 61
4.5.6 ERXADDR2, Selected Error Record Address Register 2 62
4.5.7 ERXCTLR, Selected Error Record Control Register 63
4.5.8 ERXCTLR2, Selected Error Record Control Register 2 64
4.5.9 ERXFR, Selected Error Record Feature Register 65
4.5.10 ERXFR2, Selected Error Record Feature Register 2 66
4.5.11 ERXMISC0, Selected Error Record Miscellaneous Register 0 67
4.5.12 ERXMISC1, Selected Error Record Miscellaneous Register 1 68
4.5.13 ERXMISC2, Selected Error Record Miscellaneous Register 2 69
4.5.14 ERXMISC3, Selected Error Record Miscellaneous Register 3 70

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.15 ERXSTATUS, Selected Error Record Primary Status Register 71
4.5.16 VDFSR, Virtual SError Exception Syndrome Register 72
4.5.17 VDISR, Virtual Deferred Interrupt Status Register 73
4.6 Error Record registers, including memory mapped view 74
4.6.1 Register index 74
4.6.2 ERR<*irq*>CR0, Error Interrupt Configuration Register 0 77
4.6.3 ERR<*irq*>CR1, Error Interrupt Configuration Register 1 78
4.6.4 ERR<*irq*>CR2, Error Interrupt Configuration Register 2 79
4.6.5 ERR<*n*>ADDR, Error Record Address Register 81
4.6.6 ERR<*n*>CTLR, Error Record Control Register 83
4.6.7 ERR<*n*>FR, Error Record Feature Register 88
4.6.8 ERR<*n*>MISC0, Error Record Miscellaneous Register 0 91
4.6.9 ERR<*n*>MISC1, Error Record Miscellaneous Register 1 96
4.6.10 ERR<*n*>STATUS, Error Record Primary Status Register 97
4.6.11 ERRCIDR0, Component Identification Register 0 105
4.6.12 ERRCIDR1, Component Identification Register 1 106
4.6.13 ERRCIDR2, Component Identification Register 2 107
4.6.14 ERRCIDR3, Component Identification Register 3 108
4.6.15 ERRDEVAFF, Device Affinity Register 109
4.6.16 ERRDEVARCH, Device Architecture Register 112
4.6.17 ERRDEVID, Error Record Device ID Register 114
4.6.18 ERRGSR<*n*>, Error Group Status Register 115
4.6.19 ERRIRQCR<*n*>, Generic Error Interrupt Configuration Register 116
4.6.20 ERRIRQSR, Error Interrupt Status Register 117
4.6.21 ERRPIDR0, Peripheral Identification Register 0 119
4.6.22 ERRPIDR1, Peripheral Identification Register 1 120
4.6.23 ERRPIDR2, Peripheral Identification Register 2 121
4.6.24 ERRPIDR3, Peripheral Identification Register 3 122
4.6.25 ERRPIDR4, Peripheral Identification Register 4 123

5 Appendix – Glossary 124

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Preface

This document describes the RAS Extension, and must be read in conjunction with the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

References

This document refers to the following documents.

Reference	Document number	Author	Document name
[1]	ARM DDI 0487	ARM	*ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*

Feedback

ARM welcomes feedback on its documentation.

If you have comments on the content of this specification, send an e-mail to errata@arm.com. Give:

- The title.
- The document and version number, ARM DDI 0587B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

1 Introduction

RAS are three aspects of the *dependability* of a system:

- *Reliability*, continuity of correct service.
- *Availability*, readiness for correct service.
- *Serviceability*, ability to undergo modifications and repairs.

RAS techniques reduce unplanned outages because:

- Transient errors can be detected and corrected before they cause application or system failure.
- Failing components can be identified and replaced.
- Failure can be predicted ahead-of-time to allow replacement during planned maintenance.

The RAS Extension is a mandatory extension to the ARMv8.2 architecture, and it is an optional extension to the ARMv8.0 and ARMv8.1 architectures.

ID_AA64PFR0_EL1.RAS in AArch64 state, and ID_PFR0.RAS in AArch32 state, indicate whether the RAS Extension is implemented.

The RAS Extension introduces a new barrier instruction, the Error Synchronization Barrier (ESB), which is described in detail in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

The RAS Extension also introduces a set of System and memory-mapped registers that are specific to the RAS architecture. These are described in detail in this specification.

In addition, the RAS Extension changes a limited number of fields in the following ARMv8-A System registers:

- In AArch64 state:
  - HCR_EL2.
  - ID_AA64MMFR1_EL1.
  - ID_AA64PFR0_EL1.
  - ID_MMFR4_EL1.
  - ID_PFR0_EL1.
  - SCR_EL3.
  - ESR_ELx.
  - IFSR32_EL2.
- In AArch32 state:
  - HCR2.
  - ID_MMFR4.
  - ID_PFR0.
  - SCR.
  - DFSR.
  - IFSR.
  - HSR.

For details about these changes, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile.*

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

1.1 Types of errors

Correct service is delivered when the service implements the system function. This might encompass:

- Producing correct results.
- Producing results within the time allotted to the task.
- Not divulging secret or secure information.

For the purpose of describing the RAS Extension, deviation from correct service is defined using the following terms:

- A *failure* is the event of deviation from correct service. This includes data corruption, data loss, and service loss.
- An *error* is the deviation.
- A *fault* is the cause of the error.

Errors that are present but not detected are *latent* or *undetected* errors. A transaction carrying a latent error is *corrupted*. In a system with no error detection, all errors are latent errors and are *silently propagated* by components until either:

- They are masked and do not affect the outcome of the system:
  - These are benign or false errors.
- They affect the service interface of the system and cause failure:
  - These are silent data corruptions (SDC).
  - The rate of such failures, measured as the number of failures per billion device-hours of operation, is called the SDC *Failure-in-Time* (FIT) rate.

The severity of a failure can range from minor to catastrophic:

- The harmful consequences of a *minor failure* are of a similar cost to the benefits provided by correct service delivery.
- The harmful consequences of a *catastrophic failure* are orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery.

There are many sources of faults in a system, including both software and hardware faults:

- *Hardware faults* originate in, or affect, hardware.
- *Software faults* affect software, that is programs or data.

The RAS Extension primarily addresses errors produced from hardware faults. These fall into two main areas:

- Transient faults.
- Non-transient or *persistent faults*.

1.2 Techniques for improving reliability, availability, and serviceability

Each device sets its own targets for reliability, availability, and serviceability, and uses different techniques to achieve these targets, including:

- Fault prevention and fault removal.
- Error handling and recovery.
- Fault handling.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

The RAS Extension does not prescribe the level of reliability, availability, and serviceability in any implementation, or which parts of the system include RAS.

1.2.1 Fault prevention and fault removal

Fault prevention and fault removal are two techniques for handling faults. Fault prevention and fault removal mechanisms are IMPLEMENTATION DEFINED.

Fault prevention techniques are outside the scope of the architecture.

The RAS Extension provides a common programmers' model and mechanisms for fault handling. A fault that is removed is a corrected error and might be recorded and generate a fault handling interrupt, but it is not propagated. This means that it is not consumed and does not signal an error exception.

1.2.2 Error handling and recovery

Error recovery is the process by which software and hardware minimize the impact of an uncorrected error by:

- Deferring an error from faults by not taking action immediately, but waiting for either:
  - The fault to be masked later (fault removal).
  - The error to be reported to a consumer of the error (this enables error recovery).
- Preventing further propagation of the error, that is *containing* the error.
- Reducing the severity of the error by invoking a *service failure mode*:
  - These are detected uncorrected errors (DUE).
  - The rate of such failures gives the DUE FIT rate.
  - The type of service failure mode depends on what is acceptable to the service.

A software error recovery agent is invoked when hardware detects an error it cannot correct or remove.

An error recovery agent also provides information to the operator through error logs to improve serviceability, for example to help with the identification of *field replaceable units* (FRUs).

Error detection mechanisms are IMPLEMENTATION DEFINED. The RAS Extension provides a common programmers' model and mechanisms for error recovery.

1.2.3 Fault handling

Fault handling by software is the process by which software diagnoses and responds to faults to improve availability.

The RAS Extension provides mechanisms to allow the reporting of errors and warnings to a fault handling agent, and to provide information about the fault to the agent. The detailed nature of the fault handling agent is outside the scope of this architecture

Fault handling and error recovery might be independent agents.

1.3 General taxonomy of errors

1.3.1 Error detection

When a component accesses memory or other state, an error might be detected in that memory or state. The error might be corrected, deferred, or signaled to another component as a *detected* error.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

1.3.2 Error propagation

When an error is passed from a producer to a consumer, the producer *propagates* the error. If the error is received by the consumer as an undetected error, the error has been *silently propagated*.

In the simplest case, an error is propagated if a corrupt value is passed from producer to consumer. However, an error has also been propagated if:

- A transaction that would not have been permitted to occur had the fault not been activated passes from the producer to the consumer.
- A transaction does not occur that ought to have occurred.
- Modified data is lost or any other loss of coherency in a multiprocessor coherent system is observed.

In some cases, changing the timing and order of transactions is also considered propagation of an error.

1.3.3 Infected, poisoned, containable, and uncontainable

If an uncorrected error is consumed and updates the state of the component, then that state becomes *infected*. If the state is marked as being in error, meaning a subsequent read of the state will signal a detected error, the state is *poisoned*.

An undetected error is *uncontained* at the component that failed to detect it. A detected uncorrected error is uncontained at the component that silently propagates it.

A detected uncorrected error is *uncontainable* if it might be uncontained. A detected uncorrected error is *containable* if it is not uncontainable. If the component cannot determine whether a detected error is uncontainable or containable, it must treat it as uncontainable.

An error that is uncontainable at a component might still be containable at the system level.

—— Note ——————————

Reporting an error as containable allows software to contain the error. It does not mean that hardware has contained the error.

————————————————

1.4 Fault handling

The data recorded about the error is an *error record*. It is the responsibility of the error recovery and fault handling processes to collate the error record data and write it to an *error log*. See Standard error record.

ARM recommends that the error record identifies the component where the fault originated.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

2 ARMv8-A Error Handling and Recovery Architecture

2.1 Taxonomy of errors in the ARMv8-A architecture

2.1.1 Architectural error detection

When a PE accesses memory or other state, an error might be detected in that memory or state, and corrected, deferred, or signaled to the PE as a detected error.

It is IMPLEMENTATION DEFINED whether an error detected by the consumer of a write from a PE is signaled to the PE and becomes a detected error that is consumed by the PE.

The component that detects an error is called a *node*. The nodes included as part of a processor, including an ARMv8-A PE, are IMPLEMENTATION DEFINED. An ARMv8-A PE implementing the RAS Extension must implement the System register interface to the IMPLEMENTATION DEFINED nodes of a PE. For more information, see Nodes.

The size of Protection granule for any implemented error detection mechanism is IMPLEMENTATION DEFINED. A system might implement multiple error detection mechanisms with differing Protection granule sizes. The mechanism for clearing an error or poison from a Protection granule is IMPLEMENTATION DEFINED, and it is IMPLEMENTATION DEFINED whether any such mechanism exists.

—— Note ——————————

For some systems, a single-copy atomic write of at least the whole Protection granule can reset the state of the granule and clear any error or poison. In other systems, a `DC ZVA` operation might also clear the error. However, the Protection granule might be larger than the `DC ZVA` block size and/or the largest single-copy atomic access that the PE can perform. The systems might require software to stop using the Protection granule, for example by not using the physical page containing the granule, until the system can be purged of errors, for example at a system reset. The architecture does not set any limit on the size of a Protection granule and it might be larger than a translation granule. Any mechanism for purging the system errors is also IMPLEMENTATION DEFINED.

————————————————

2.1.2 Architectural error propagation

For a PE, Error propagation applies to the propagation of detected errors between the general-purpose or SIMD&FP registers, and any program-visible architectural state of the PE, including:

- Other general-purpose and SIMD&FP registers.
- System registers.
- Special-purpose registers.
- PSTATE.
- Memory.

That is, the error is propagated by:

- A store of a corrupt value.
- A write of a corrupt value to a System register, Special-purpose register or PSTATE. Infecting a System register state might mean that the PE generates transactions that would not otherwise be permitted.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- Any operation that would not have been permitted to occur had the error not been activated, including:
  - A load, translation table walk, or instruction fetch that would not have been permitted, including those from hardware speculation or prefetching.
  - A store to an incorrect address or a store that would not have been made or not permitted.
  - A direct or indirect write to a Special-purpose or System register that would not have been made or not permitted.
  - Assertion of any signal, such as an interrupt, that would not have been asserted.
- Any operation not occurring that would have occurred had the error not been activated.
- Taking an imprecise exception.
- The PE discarding data that it holds in a modified state.
- Any other loss of uniprocessor semantics, ordering, or coherency.

The propagated error is silently propagated if it is not signaled to the consumer as a detected error.

The features that a PE includes to contain an error are IMPLEMENTATION DEFINED, and it is IMPLEMENTATION DEFINED whether an error can be signaled to the consumer as a detected error.

For example, an implementation might ensure that a corrupt value in a general-purpose or SIMD&FP register is not silently propagated, by signaling an error on any write of corrupt data to a memory location so that the memory location is poisoned.

2.1.3 Architecturally infected, containable, and uncontainable

Infected, poisoned, containable, and uncontainable apply to all program-visible architectural state of the PE, including general-purpose registers, SIMD&FP registers, special-purpose and System registers, PSTATE, and memory.

An error is uncontained by the PE if the error is silently propagated, unless it is contained because all of the following are true:

- The corrupt value is in the general-purpose or SIMD&FP registers.
- The error is only silently propagated by an instruction that occurs in program order after one of the following:
  - Taking the External Abort or SError interrupt exception generated by the error.
  - An Error Synchronization Barrier operation that synchronizes the error.
- The error is not silently propagated in any other way.

Architectural error propagation defines error propagation for a PE.

2.1.4 Architecturally consumed errors

For a PE, an error is architecturally consumed if any of the following are true:

- An instruction commits the corruption into the visible state of the PE.
- The error is on an instruction fetch and the instruction is committed for execution.
- The error is on a translation table walk for a committed load, store, or instruction fetch.

The PE must take action for a detected, architecturally consumed error, either by:

- Generating an error exception.
- Entering a failure mode.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

2.1.5 Other errors

Errors from software faults are outside the scope of the RAS Extension error recovery architecture.

From within the processor itself, other errors might be detected. These are not errors detected by the architectural model of the PE and so are treated like errors detected by another component. An example of this is when the cache, not the PE, detects a tag RAM error. Other components might report errors to a PE using error recovery interrupts.

For implementations that include the Statistical Profiling Extension, the Statistical Profiling Extension behaves like a separate component.

2.2 Generating error exceptions

2.2.1 Error correction and deferment

If hardware can correct or defer a detected error, it must do so. The error is logged, and a fault handling interrupt is generated for fault handling purposes.

2.2.2 Error exceptions

An error exception might be generated for an error that is corrected or deferred.

An error exception must be generated for all detected errors that are neither corrected nor deferred, and are signaled to and consumed by a PE as an external abort in response to:

- An architectural read from memory.
- A write to memory or a data cache maintenance operation. It is IMPLEMENTATION DEFINED whether an error detected by the consumer of a write from a PE:
  - Is deferred to the consumer, for example, by poisoning the location.
  - Is signaled to the PE as an external abort.
  - Generates an error recovery interrupt.

It is IMPLEMENTATION DEFINED whether an error that is consumed by hardware speculation or prefetching by a PE, but that is not committed to the architecturally visible state of the PE, generates an external abort exception at the PE. If an external abort is generated, it must be taken asynchronously as an SError interrupt.

—— Note ——————————

An SError interrupt can also be generated for IMPLEMENTATION DEFINED causes.

————————————————

It is IMPLEMENTATION DEFINED whether an external abort is taken as:

- A synchronous external abort exception.
- An asynchronous external abort that is taken as an *SError interrupt* exception.

On each error exception, it is IMPLEMENTATION DEFINED whether the error has been contained or is *Uncontainable*. If the error has been contained, it is further IMPLEMENTATION DEFINED whether the state of the PE on taking the error exception is *Unrecoverable*, *Recoverable*, or *Restartable*:

Uncontainable error (UC)

The error is Uncontainable if the error has been, or might have been, silently propagated.

If the error cannot be isolated to an application or VM, or both, the system must be shut down by software to avoid catastrophic failure.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Unrecoverable error (UEU)

The state of the PE is Unrecoverable if all of the following are true:

— The error has not been silently propagated.

— The PE cannot recover execution from the preferred return address of the exception. This might be because of one of the following:

- The error has been architecturally consumed by the PE and infected the state of the PE general-purpose, SIMD&FP, and System registers.
- The exception is imprecise.

Either the application or the VM, or both, cannot continue and must be isolated by software.

Recoverable error (UER)

The state of the PE is Recoverable if all of the following are true:

— The error has not been silently propagated.

— The error has not been architecturally consumed by the PE. (The PE architectural state is not infected.)

— The exception is precise and PE can recover execution from the preferred return address of the exception, if software locates and repairs the error.

The PE cannot make correct progress without either consuming the error or otherwise making the error unrecoverable. The error remains latent in the system.

If software cannot locate and repair the error, either the application or the VM, or both, must be isolated by software.

Restartable error (UEO)

The state of the PE is Restartable if all of the following are true:

— The error has not been silently propagated.

— The error has not been architecturally consumed by the PE. (The PE architectural state is not infected.)

— The exception is precise and PE can recover execution from the preferred return address of the exception without any immediate action by software.

The PE can make progress. However, the error remains latent in the system.

Software might take action to locate and repair the error before it is consumed. The PE can be restarted by software without software taking any action to locate and repair the error.

This taxonomy is shown by Figure 1.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

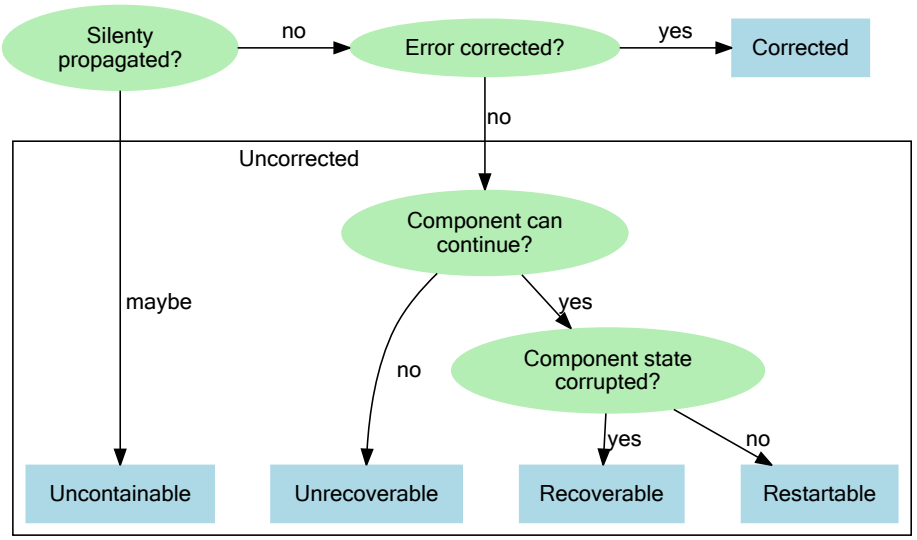


Figure 1: Taxonomy of consumed error types

The severity of the error and the state of the PE are reported when the error exception is taken. See:

- AArch64 syndrome registers on taking an error exception.
- AArch32 syndrome registers on taking an error exception.

The set of error types that can be reported by an implementation is IMPLEMENTATION DEFINED. An implementation can report:

- Any Restartable error as any of Recoverable, Unrecoverable, or Uncontainable.
- Any Recoverable error as either Unrecoverable or Uncontainable.
- Any Unrecoverable error as Uncontainable.

Note

If the state of the PE is reported as Recoverable, this does not mean that the error can be recovered from because the error in memory might be one which does not allow software to recover the operation. Rather, software *might* be able to recover if it can repair the error and continue.

2.3 Error record System register view

Note

AArch64 Error record System registers are all registers with an ER*_EL1 mnemonic. AArch32 Error record System registers are all registers with an ER* mnemonic. Both of these are defined in this specification.

The number of error records that can be accessed through the System registers is IMPLEMENTATION DEFINED, and might be zero. The ERRIDR_EL1 and ERRIDR registers indicate the highest numbered index of the error records that can be accessed by System registers, plus one.

The error record System registers are described by:

- AArch64 System registers.
- AArch32 System registers.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

The error record register contents are described by Error Record registers, including memory mapped view.

To access an error record, software must:

- Set the error selection register, ERRSELR_EL1.SEL or ERRSELR.SEL, to the index of the record being accessed.
- Access the error record using the ERX* System registers.

—— Note ——————————

See Synchronization and error record accesses.

————————————————

The error records accessed through the System registers might be accessible only to the PE associated with those Systems registers, or they might be shared and therefore accessible to other PEs through either System registers or as a memory-mapped component.

2.3.1 Fields in VSESR_EL2, VDFSR, DISR(_EL1), and VDISR(_EL2)

ESR_ELx, HSR, DFSR, VSESR_EL2, VDFSR, DISR_EL1, DISR, VDISR_EL2, and VDISR are error syndrome registers that are written with either a syndrome by hardware on taking or deferring a physical SError interrupt, or with a virtual syndrome value provided by software for a virtual SError interrupt, as applicable.

For a given implementation:

- If ESB never synchronizes any errors, then DISR_EL1.A and DISR.A might be RES0.
- The error syndrome registers must be capable of storing any syndrome value that might be reported by hardware on taking a physical error exception.
- If any of ESR_ELx[24:0], HSR[11:9], and DFSR[15:14,12] is not used and always set to zero by hardware on taking a physical SError interrupt exception or synchronous External Abort exception, it can be RES0 in that syndrome register.
- A bit that is not used and always set to zero or always set to one by hardware on taking a physical SError interrupt is permitted to be RES0 or RES1 respectively in the corresponding other syndrome registers. See Table 1.

Table 1: Permitted relaxations for bits in error syndrome registers

Bit that is permitted to be RES0 or RES1	**If always set to zero or always set to one on taking an SError interrupt in all of**		
	ESR_ELx[*x*], *x* ∈ [24:0]	HSR[*x*], *x* ∈ [11:9]	DFSR[*x*], *x* ∈ [15:14,12]
VSESR_EL2[*x*]	Yes	-	Yes
VDISR_EL2[*x*]	Yes	-	Yes
DISR_EL1[*x*]	Yes	-	-
VDFSR[*x*]	-	-	Yes
VDISR[*x*]	-	-	Yes
DISR[*x*]	-	Yes	Yes

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

2.4 Error Synchronization Barrier

The RAS Extension adds the Error Synchronization Barrier operation and the ESB instruction. For details and the encoding of ESB, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile.*

In AArch32 state:

- ESB might update the DISR or VDISR registers.
- The DISR register can only be accessed at EL1 or above. If EL2 is implemented and HCR.AMO is set to 1, then reads and writes of DISR at Non-secure EL1 access VDISR.

In AArch64 state:

- ESB might update the DISR_EL1 or VDISR_EL2 registers.
- The DISR_EL1 register can only be accessed at EL1 or above. If EL2 is implemented and HCR_EL2.AMO is set to 1, then reads and writes of DISR_EL1 at Non-secure EL1 access VDISR_EL2.

Error Synchronization Barriers synchronize Unrecoverable errors, that is, containable errors that are architecturally consumed by the PE and not silently propagated.

All Unrecoverable errors must be synchronized by Error Synchronization Barriers.

2.4.1 ESB and Unrecoverable errors

For Unrecoverable errors, an ESB guarantees that:

- All Unrecoverable errors generated in program order before the ESB have pended a physical SError interrupt exception.
- If such a physical SError interrupt is pended by the ESB instruction, or was pending before the ESB instruction is executed:
  - If the physical SError interrupt is unmasked at the current Exception level, it is taken before completion of the ESB instruction.

    —— Note ——————————

    - That is, the preferred return address of the exception is the ESB instruction address.
    - The prioritization of this exception with respect to any other asynchronous exceptions that are pending and unmasked when the ESB instruction is executed is IMPLEMENTATION DEFINED.

    ————————————————

  - If the physical SError interrupt is masked at the current Exception level, then:
    - The pending SError interrupt is cleared.
    - The SError interrupt syndrome is recorded in DISR_EL1 or DISR.
    - The DISR_EL1.A or DISR.A field is set to 1 to indicate the SError interrupt was generated in program order before the ESB.

The Error Synchronization Barrier operation contains the error. See Architecturally infected, containable, and uncontainable.

ESB guarantees that all Unrecoverable SError interrupts that are generated in program order:

- Before the ESB, are either taken before or at the ESB, or recorded in the DISR_EL1 or DISR register by the ESB.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- After the ESB, are taken after the ESB and not recorded in DISR_EL1 or DISR by the ESB.

This includes Unrecoverable SError interrupts that are generated by instructions, translation table walks, hardware updates to the translation tables, and instruction fetches on the same PE.

If the Unrecoverable SError interrupt exception is:

- Taken, then the error is reported to software as Unrecoverable in the ESR_ELx, DFSR or HSR syndrome register on taking the interrupt.
- Not taken, then the error is reported to software as Unrecoverable in the DISR_EL1 or DISR register.

2.4.2 ESB and other containable errors

For other types of containable error:

- A *Recoverable* error has not yet been consumed by the PE.
- *Restartable* and *Corrected* errors, and SError interrupts from reads by hardware speculation that do not corrupt the state of the PE, have not been consumed by the PE.

An unconsumed SError interrupt taken at the ESB or recorded in the DISR_EL1 or DISR register, might have been generated by hardware speculation of an instruction in program order after the Error Synchronization Barrier.

These SError interrupts are always precise.

2.4.3 ESB and other physical errors

For other errors, the following rules apply:

- Synchronous Abort exceptions are not synchronized by an Error Synchronization Barrier.
- Error recovery interrupts are asynchronous and are not synchronized by an Error Synchronization Barrier.
- It is IMPLEMENTATION DEFINED whether IMPLEMENTATION DEFINED and uncategorized SError interrupts are containable or Uncontainable, and whether they can be synchronized by an Error Synchronization Barrier.
- Uncontainable errors might not have been contained, and Uncontainable SError interrupts might be imprecise. An Uncontainable error might be taken at the ESB or recorded in the DISR_EL1 or DISR register by an ESB instruction, but this is not architecturally required.

2.4.4 ESB and virtual errors

If EL2 is implemented, then an ESB instruction executed at Non-secure EL0 or EL1 also synchronizes a pending virtual SError interrupt when any of:

- EL2 is using AArch64, HCR_EL2.AMO is set to 1, and HCR_EL2.TGE is set to 0.
- EL2 is using AArch32, HCR.AMO is set to 1, and HCR.TGE is set to 0.

In these cases, if a virtual SError interrupt is pending when the ESB instruction is executed:

- If the virtual SError interrupt is unmasked at the current Exception level, it is taken before the completion of the ESB instruction.
- If the virtual SError interrupt is masked at the current Exception level:
    - HCR_EL2.VSE or HCR.VA is cleared to 0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

— The virtual SError interrupt syndrome from VSESR_EL2 or VDFSR is recorded in VDISR_EL2 or VDISR, and VDISR_EL2.A or VDISR.A is set to 1 to indicate the SError interrupt was pending prior to the execution of the ESB instruction.

—— Note ——————————

This happens in parallel with the Error Synchronization Barrier operation for physical SError interrupts.

————————————————

2.4.5 Extension for barrier at exception entry and exit

The ARMv8.2 IESB architectural feature adds a control bit to each SCTLR_ELx register to insert an *implicit* Error Synchronization Barrier operation at exception entry and exception return. For the register field descriptions, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile.*

For each Exception level, ELx, other than EL0, the implicit Error Synchronization Barrier after each exception or DCPSx instruction that is taken to ELx guarantees that Unrecoverable errors that are generated before the exception have pended an SError interrupt exception. If an Unrecoverable SError interrupt that can be synchronized is pended by, or was pending before the exception, then:

- If SError interrupts are unmasked at ELx, the SError interrupt is taken before the first instruction at ELx is executed.

  —— Note ——————————

  That is, the preferred return address for the SError interrupt is the original exception vector address.

  ————————————————

- If SError interrupts are masked at ELx, the SError interrupt is pending.

The implicit Error Synchronization Barrier before an ERET or DRPS instruction executed at ELx guarantees that Unrecoverable errors that are generated before the return instruction have pended an SError interrupt exception. If an Unrecoverable SError interrupt is pended by, or was pending before the instruction, then:

- If SError interrupts are unmasked, the SError interrupt is taken before the return instruction.

  —— Note ——————————

  That is, the preferred return address for the SError interrupt is the address of the return instruction.

  ————————————————

- If SError interrupts are masked, the SError interrupt is pending.

The implicit form of Error Synchronization Barrier:

- Must synchronize all Unrecoverable errors.
- Has no effect on DISR_EL1 or VDISR_EL2.
- On taking an SError interrupt to ELy, reports that the error was generated by an implicit Error Synchronization Barrier in the ESR_ELy.IESB bit.
- Guarantees that:
  - — All Unrecoverable errors generated in program order before the implicit barrier, are taken at the implicit barrier if the SError exception is unmasked.
  - — No Unrecoverable errors generated in program order after the implicit barrier are reported in this way.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Note

The prioritization of asynchronous interrupts is IMPLEMENTATION DEFINED. This means that an implementation might choose to:

- Behave as if the SError interrupt was taken before the implicit Error Synchronization Barrier operation, if the interrupt was not masked.
  - For the exception entry case, taking the SError interrupt in place of the exception. The syndrome information for the SError interrupt (including the restart address and whether the PE state is Restartable, Recoverable, or otherwise) must then apply to the instruction that generated the exception.
  - For the exception return case, the ESR_ELy.IESB bit might be zero, but, otherwise the behavior is the same.
- Take another exception before the SError interrupt. In this case, the SError interrupt remains pending.

ARM recommends the SError interrupt is prioritized over other exceptions.

2.5 Taking external abort and SError interrupt exceptions to AArch64

2.5.1 Target Exception level

For SError interrupt and synchronous external abort exceptions taken to AArch64 state, the default target Exception level is:

- EL1, if taken from EL0 or EL1.
- EL2, if taken from EL2.
- EL3, if taken from EL3.

However:

- If EL3 is implemented and SCR_EL3.EA is set to 1, all SError interrupt and synchronous external abort exceptions are taken to EL3.
- Otherwise, if EL2 is implemented and the PE is in Non-secure state, then:
  - If HCR_EL2.AMO or HCR_EL2.TGE is set to 1, all SError interrupts from EL0 and EL1 are taken to EL2.
  - If HCR_EL2.TEA or HCR_EL2.TGE is set to 1, all synchronous external abort exceptions from EL0 and EL1 are taken to EL2.

2.5.2 AArch64 syndrome registers on taking an error exception

On taking an SError interrupt or synchronous external abort exception to an Exception level using AArch64, the error syndrome is recorded in ESR_ELx.

2.5.3 Multiple SError interrupts

An SError interrupt is asynchronous and might be masked by PSTATE.A. Therefore, multiple SError interrupt conditions might be pending together. The architecture does not define relative priorities for asynchronous exceptions.

It is IMPLEMENTATION DEFINED whether the multiple pending SError interrupt conditions are taken as a single SError interrupt exception.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

The value of ESR_ELx is IMPLEMENTATION DEFINED if the multiple pending SError interrupt conditions that are taken as a single SError interrupt exception comprise a mix of:

- Pending IMPLEMENTATION DEFINED SError interrupt conditions that would be reported with ESR_ELx.IDS == 1.
- Pending SError interrupt conditions that would be reported with ESR_ELx.IDS == 0.

On taking an SError interrupt exception for one or more SError interrupt conditions:

- If one or more pending SError interrupt conditions are reported with ESR_ELx.IDS == 1, then the syndrome recorded in ESR_ELx.ISS is IMPLEMENTATION DEFINED.
- If all pending SError interrupt conditions are reported with ESR_ELx.IDS == 0, then the combined effect of the errors on the state of the PE is reported in ESR_ELx.AET.

Any pending SError interrupt conditions that are not taken with other SError interrupts as a single SError interrupt exception remain pending in ISR_EL1 after the SError interrupt exception is taken.

An Error Synchronization Barrier operation requires that all Unrecoverable errors must be synchronized. If there are multiple requests outstanding, they are all synchronized by a single Error Synchronization Barrier operation.

2.6 Taking external abort and SError interrupt exceptions to AArch32

2.6.1 Target mode

For SError interrupt and synchronous external abort exceptions taken to AArch32 state, the default target mode is:

- Abort mode, if taken from EL0, EL1 or EL3, including from Secure Monitor mode.
- Hyp mode, if taken from EL2.

However:

- If EL3 is implemented and using AArch32 and SCR.EA is set to 1, all SError interrupt and synchronous external abort exceptions are taken to Secure Monitor mode, using vector offset `0x10`.
- Otherwise, if EL2 is implemented and using AArch32 and is in Non-secure state:
  - If HCR.AMO or HCR.TGE is set to 1, all SError interrupts from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.
  - If HCR.TEA or HCR.TGE is set to 1, all synchronous external abort exceptions from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.

2.6.2 AArch32 syndrome registers on taking an error exception

On taking an SError interrupt or synchronous external abort exception to an Exception level using AArch32, the error syndrome is recorded in:

- DFSR if an SError interrupt or synchronous Data Abort is taken to a PL1 mode.
- IFSR if a synchronous Prefetch Abort is taken to a PL1 mode.
- HSR if taken to Hyp mode.

2.6.3 Multiple SError interrupts

An SError interrupt is asynchronous and might be masked by CPSR.A. Therefore, multiple SError interrupt conditions might be pending together. The architecture does not define relative priorities for asynchronous exceptions.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

It is IMPLEMENTATION DEFINED whether the multiple pending SError interrupt conditions are taken as a single SError interrupt exception.

On taking an SError interrupt exception, whether for one or more SError interrupt conditions, the combined effect of the errors on the state of the PE is reported in HSR.AET or DFSR.AET.

Any pending SError interrupt conditions that are not taken with other SError interrupts as a single SError interrupt exception remain pending in ISR after the SError interrupt exception is taken.

An Error Synchronization Barrier operation requires that all Unrecoverable errors must be synchronized. If there are multiple requests outstanding, they are all synchronized by a single Error Synchronization Barrier operation.

2.7 Virtual SError interrupts

When implemented, EL2 provides a virtual SError interrupt. The RAS Extension provides:

- Virtual syndrome registers to allow the ESR_EL1 or DFSR value to be specified on taking a virtual SError interrupt. See VSESR_EL2 and VDFSR.
- Support for EL0 or EL1 to isolate a virtual SError interrupt as if it were a physical SError interrupt. See ESB and virtual errors.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

3 RAS System Architecture

3.1 Recording errors

The component that detects an error is called a *node*. See Nodes.

When an error is detected by a node that supports fault reporting, it records the error and might generate a *fault handling interrupt* that is usually sent to an interrupt controller.

ARM recommends that nodes record all errors in *error records* to enable error recovery and fault handling. However, it is IMPLEMENTATION DEFINED whether:

- Errors are recorded by one or more of the following:
  - The node that detects them.
  - The master that consumes a signaled detected error. If only the master records the error, then syndrome information must be passed with the error to the master:
  - A third party, that is a node whose purpose is to record errors for other nodes. Typically, such a node contains either one record for each node for which it is recording an error, or it uses the error record to identify the originating node.
- A master that consumes a signaled detected error records the consumed error.
- Errors are recorded by a node that merely propagates a signaled detected error from a different master to a different consumer.
- All corrected errors are recorded.
- Errors detected on hardware speculation are recorded.

An error record holds syndrome for the error. It contains:

- The status of the error.
- An address, if applicable.
- Optionally, counters for software to poll the rate of Corrected errors.
- Information to identify a FRU and locate the error within the FRU.
  - This information is IMPLEMENTATION DEFINED.
- Other IMPLEMENTATION DEFINED information.

The Error Record registers might also contain control registers for error detection, correction and reporting at the node.

3.1.1 Detected uncorrected errors in data

Detected uncorrected errors are recorded and a fault handling interrupt is raised when the error is detected. ARM recommends that the hardware records enough information to allow fault analysis to find trends in the faults and allow identification of FRUs.

3.1.2 Corrected errors in data

ARM recommends that corrected errors are recorded by counting the individual errors. The fault handling process might compare the corrected error rate with a threshold value to determine whether to take action.

ARM recommends that hardware also records enough information to:

- Allow fault analysis to find trends in the faults. This information is IMPLEMENTATION DEFINED but might include the location of the data.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- Allow identification of a FRU.

The error is corrected and the corrected data passed on to the consumer.

It is IMPLEMENTATION DEFINED whether counting is done by software or hardware. Standard format Corrected error counter and Corrected error counting describe an optional standard hardware mechanism for counting errors.

3.1.3 Other sources of error and warnings

Other sources of error and warning are possible in a system. Within the ARMv8-A architecture these are signaled to a PE using an error recovery or fault handling interrupt.

3.1.4 Error types

For a Standard error record, the types of error that can be recorded are:

Corrected error

The error was detected and corrected. The error no longer infects the state of the node and has not been silently propagated. The node continues to operate.

Deferred error

The error was detected, was not corrected, and was deferred. The error has not been silently propagated. The error might be latent in the system. It is IMPLEMENTATION DEFINED whether the error continues to infect the state of the node or whether it has been deferred to the consumer. The node continues to operate. If the error might have been silently propagated, it must be reported as an Uncorrected error.

Uncorrected error

The error was detected and was not corrected or deferred. The error is latent in the system. An Uncorrected error can have sub-types. Two common sub-types are:

- *Unrecoverable*: The error has not been silently propagated.
- *Uncontainable*: The error might have been silently propagated. If the error cannot be isolated, the system must be shut down to avoid catastrophic failure.

The following two sections list additional sub-types.

This taxonomy is shown by Figure 2.

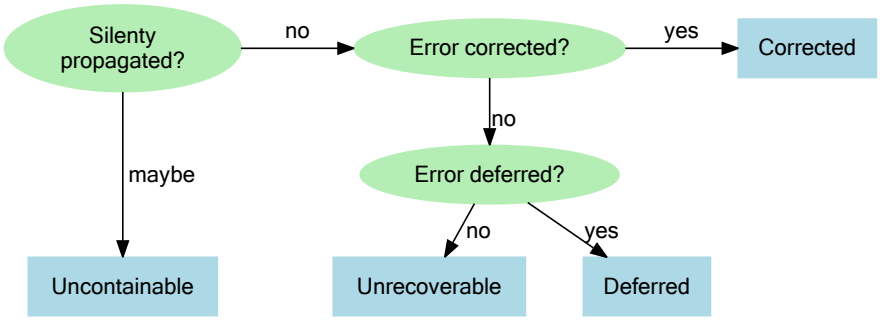


Figure 2: Taxonomy of producer error types

Additional subtypes for an Uncorrected error produced at the node

For an error produced at the node, a node might additionally define the following subtypes:

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Latent

The error has not been propagated. That is, the error was detected but not consumed, and was not recorded as a deferred error.

Signaled

The error has not been silently propagated. The error has been or might have been consumed, and was not recorded as a deferred error.

—— **Note** ——————————

The producer does not know if a consumer has architecturally consumed the error. An error might be marked as latent if it has definitely not been propagated to any consumer, and signaled otherwise.

————————————————

If these subtypes are implemented, then the Unrecoverable subtype further indicates that the node cannot continue operating.

ARM deprecates the implementation of the Latent and Signaled Uncorrected error types in the RAS system architecture.

Additional subtypes for an Uncorrected error consumed at the node

For an error consumed at the node, a node might additionally define the following subtypes:

Restartable

The error has not been silently propagated. The node has halted operation because it has consumed an error. The node does not rely on the corrupted data, so it can continue to operate without repairing the error.

Recoverable

The error has not been silently propagated. The node has halted operation. The node is reliant on consuming the corrupted data to continue. If software can locate and repair the error, the halted operation can continue.

If these subtypes are implemented, then the Unrecoverable subtype further indicates that the node cannot resume from its halted state.

ARM deprecates the implementation of the Restartable and Recoverable UE types for nodes other than consumer nodes with PE-like behavior. That is, nodes that process data that can support the concept of being restarted after halting because of an Uncorrected error.

3.1.5 Software faults

Software fault handling is outside the scope of the RAS Extension. ARM recommends that components raise an error recovery interrupt in response to a detected software fault.

3.2 Standard error record

The RAS Extension defines a standard error record and a mechanism to access error records as System registers or as a memory-mapped component. The standard error record contains:

- The ERR<n>CTLR register to control common features, and an identification mechanism for these controls.
- A status register, ERR<n>STATUS, for common status fields, such as the type and coarse characterization of the error.
- An optional address register, ERR<n>ADDR.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- An identification register, ERR<n>FR, that describes the implemented features.
- IMPLEMENTATION DEFINED status registers, ERR<n>MISC0 and ERR<n>MISC1. ARM recommends these are used for:
  - Identifying a FRU.
  - Locating the error within the FRU.
  - Optional Corrected Error counters for software to poll the rate of Corrected errors.

An error record can also include additional IMPLEMENTATION DEFINED controls and identification registers.

Error records are accessed through groups of error records, corresponding to one or more nodes. This group might be sparse and error records that are not implemented in a group of error records have an ERR<n>FR register that reads as zero.

Error records can be accessed using System registers or as a memory-mapped component:

- Error record System register view defines an architectural format for a group of error records, accessed using System registers.
- Memory-mapped view defines a reusable format for a memory-mapped group of error records.
  - Use of the reusable format is OPTIONAL.

Error records must be preserved over Error Recovery reset. This allows for a diagnosis after system failure.

3.2.1 Nodes

The architecture defines the following common features for a node:

Error detection and correction

The level of error correction and detection implemented at a node is IMPLEMENTATION DEFINED. A node might include a control to disable error reporting and logging of detected errors, for example while software initializes the node. It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting and logging are disabled.

Fault handling interrupt

This is the asynchronous reporting of all or some detected errors by an interrupt, that is, all Corrected errors, Deferred errors, and Uncorrected errors. It is IMPLEMENTATION DEFINED whether a node provides a single control for all errors, or a first control for Corrected errors and a second control for all other detected errors.

See also Fault handling interrupt and RAS System Architecture.

Corrected error counting

It is IMPLEMENTATION DEFINED whether a node that implements error correction implements a counter for counting Corrected errors. Software might poll the error counter or initialize the counter with a (negative) threshold value and receive an interrupt when the counter overflows.

It is IMPLEMENTATION DEFINED which Corrected errors are counted.

It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted by the Corrected error counter.

See also Standard format Corrected error counter.

In-band Uncorrected error signaling (external aborts)

This is the in-band signaling of detected Uncorrected errors to the consumer of the error. It is also referred to as an external abort. Corrected errors and Deferred errors are not reported by such means.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Uncorrected error recovery and handling interrupt

This is the asynchronous (out-of-band) reporting of detected Uncorrected errors by an interrupt. The interrupt can be used for error recovery, fault handling, or both. Corrected errors are not reported by this means. It is IMPLEMENTATION DEFINED whether the node provides a control to enable Deferred errors to be reported in this way. If the control is not provided, Deferred errors are not reported by this means.

The error recovery interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

Records

A node implements one or more records. When an error is detected, syndrome about the error is written to a record.

A node might implement some or all of these features, and, if implemented, might implement controls to enable or disable the features in ERR<n>CTLR.

For each node, it is IMPLEMENTATION DEFINED whether the fault and error reporting mechanisms apply to both reads and writes, or whether the mechanisms can be individually controlled for reads and writes.

3.2.2 Multiple records per node

Each node contains at least one error record. A node might implement multiple error records for one or more of the following purposes:

- To record different types of error in different records.
- To record errors from different sources in different records.
- To record multiple errors.

The multiple error records used by a single node are sequentially indexed. Each record other than the first record has an ERR<n>FR register that reads as zero, and the value in ERR<n>CTLR for each record other than the first record is RES0.

3.2.3 Writing the error record

When a new error is detected, the node:

- Modifies ERR<n>STATUS.{CE, DE, UE, UET} to indicate the type of the new detected error. See Prioritizing errors.
- Does one of the following:
  - Overwrites the error record with the syndrome for the new error, if it is of a higher priority than the previous highest priority recorded error.
  - Keeps the syndrome for the previous error, if the new error is of a lower priority than the previous highest priority recorded error or if it has the same priority as the previous highest priority recorded error.
- Counts the error, if it is a Corrected Error and a counter is implemented. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted by the Corrected error counter.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Prioritizing errors

The highest priority recorded error type is recorded in the ERR<n>STATUS.{CE, DE, UE, UET} fields, as shown in Table 2.

Table 2: Encoding the highest priority error

ERR<n>STATUS fields						
V	**CE**	**DE**	**UE**	**UET**	**Highest priority error type**	**Mnemonic**
0	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	None	-
1	`0b00`	0	0	UNKNOWN		
1	!= `0b00`	0	0	UNKNOWN	Corrected error	CE
1	X	1	0	UNKNOWN	Deferred error	DE
1	X	X	1	`0b10`	Uncorrected: Latent or Restartable error	UEO
1	X	X	1	`0b11`	Uncorrected: Signaled or Recoverable error	UER
1	X	X	1	`0b01`	Uncorrected: Unrecoverable error	UEU
1	X	X	1	`0b00`	Uncorrected: Uncontainable error	UC

Overwriting depends on the type of the previous highest priority error and on the type of the newly recorded error, as shown in Table 3.

In Table 3, the row and column headings use the mnemonics from Table 2 and the following additional abbreviations are used:

K Keep. Keep the previous error syndrome. It is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 1 or unchanged.

O Overflow. Keep the previous error syndrome and set ERR<n>STATUS.OF to 1.

W Overwrite. Record the new error syndrome. It is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0 or unchanged.

CK Count and keep. Count CE if a counter is implemented, and keep the previous error syndrome. If the counter overflows, or if no counter is implemented, it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 1 or unchanged.

CWK Count and overwrite or keep. The behavior is IMPLEMENTATION DEFINED and described by the value of ERR<n>FR.CEO:

- **0** Count CE if a counter is implemented. Keep the previous error syndrome. If the counter overflows, or if no counter is implemented, ERR<n>STATUS.OF is set to 1.
- **1** Count CE. If ERR<n>STATUS.OF == 1 before the CE is counted, keep the previous syndrome. Otherwise record the new error syndrome.

If the counter overflows, or if no counter is implemented, ERR<n>STATUS.OF is set to 1.

CW Count and overwrite. Count CE if a counter is implemented, and overwrite. If a counter is implemented and overflows, ERR<n>STATUS.OF is set to an UNKNOWN value. Otherwise, it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0 or unchanged.

WO Overwrite and overflow. ERR<n>STATUS.OF is set to 1.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

—— Note ——————————

One of ERR<n>STATUS.{CE, DE, UE} is set or modified even if the new error syndrome is discarded.

————————————————

It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted by the Corrected error counter. If counting a Deferred or Uncorrected error causes the counter to overflow, then ERR<n>STATUS.OF is set as it would be for a Corrected error that causes overflow. However, if the architecture requires that recording a Deferred or Uncorrected error sets the ERR<n>STATUS.OF flag, then this flag must be set even if the error is counted and the counter does not overflow.

Table 3: Rules for overwriting error records

Previous error type	New detected error type					
	CE	**DE**	**UEO**	**UER**	**UEU**	**UC**
-	CW	W	W	W	W	W
CE	CWK	W	W	W	W	W
DE	CK	O	W	W	W	W
UEO	CK	K	O	WO	WO	WO
UER	CK	K	O	O	WO	WO
UEU	CK	K	O	O	O	WO
UC	CK	K	O	O	O	O

Overwriting the error syndrome

When an old record is overwritten:

- Bits shown as X in Table 2 for the new error are unchanged.
- The ERR<n>STATUS.{ER, PN, IERR, SERR} syndrome fields are written with the syndrome for the new error.
- If there is an address syndrome for the new error, ERR<n>STATUS.AV is set to 1 and the address is written to ERR<n>ADDR. Otherwise ERR<n>STATUS.AV is set to 0.
- If there is other miscellaneous syndrome for the new error, it is written to the ERR<n>MISC0 or ERR<n>MISC1 registers and ERR<n>STATUS.MV is set to 1.
- If there is no additional miscellaneous syndrome for the new error written to the ERR<n>MISC0 or ERR<n>MISC1 registers, then it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.MV is set to 0 or unchanged.
  - — If software can determine from the ERR<n>MISC0 and ERR<n>MISC1 contents that the syndrome is not related to the highest priority error, the MV bit is unchanged.
  - — Otherwise the MV bit is cleared to zero.
- ERR<n>STATUS.V is set to 1.

Keeping the previous error syndrome

When an old record is kept:

- ERR<n>STATUS.{PN, IERR, SERR},ERR<n>ADDR, and ERR<n>STATUS.AV are unchanged. If the new error is an Uncorrected error, then ERR<n>STATUS.UET is unchanged.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- It is IMPLEMENTATION DEFINED whether one or both of ERR<n>MISC0 and ERR<n>MISC1 are updated. The contents of the ERR<n>MISC0 and ERR<n>MISC1 registers are IMPLEMENTATION DEFINED. Therefore, it is possible that some of the information about an otherwise discarded error is recorded in these registers. If data is written to ERR<n>MISC0 or ERR<n>MISC1 then ERR<n>STATUS.MV is set to 1.

The applicable one of ERR<n>STATUS.{CE, DE, UE} is set or modified even if the new error syndrome is discarded. DE and UE are single-bit fields that are set to 1. CE is a 2-bit field that might be modified.

Detecting multiple errors

If a node detects multiple errors simultaneously, it is IMPLEMENTATION DEFINED whether the node behaves:

- As if all errors were recorded, in any order. In this case, the prioritization rules mean that the highest priority error is recorded in the syndrome registers. However, the final value of the syndrome registers might depend on the logical order in which the errors were recorded.
- As if the highest priority error was recorded and one or more of the lower priority errors were not recorded.

If multiple errors are corrected simultaneously, and a Corrected error counter is implemented, it is IMPLEMENTATION DEFINED whether all the Corrected errors are counted.

The error types implemented at a node

The error types implemented at a node are IMPLEMENTATION DEFINED. An implementation might only include a simplified subset of these error types. A node can always record:

- UEO as any of UER, UEU, or UC.
- UER as either UEU or UC.

3.2.4 Security and Virtualization

Access to the Error System register view of error record registers can be controlled using Trap exceptions. See the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

If a component implementing Error System register access to error records processes Secure data, then either:

- Software must configure the Trap exception controls to prevent Non-secure access to the error records.
- The component provides reduced functionality to Non-secure state that does not affect operation in Secure state, or does not provide visibility of Secure data, or both.

If a memory-mapped component processes Secure data, then the error records must either:

- Be visible only to Secure accesses.
- Provide reduced visibility to Non-secure accesses that does not affect operation in Secure state, or not provide visibility of Secure data, or both.

If a memory-mapped component processes only Non-secure data, then it is IMPLEMENTATION DEFINED whether:

- The error records are visible to both Non-secure and Secure accesses.
- It is configurable whether the error records are visible to Non-secure accesses.
- The error records are visible only to Secure accesses.

If the memory-mapped component includes registers to generate *message signaled interrupts* (MSIs) and the component can be programmed by Non-secure accesses, the MSIs must not target Secure addresses.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

3.2.5 Synchronization and error record accesses

When a node detects an error it updates the Error Record registers and might generate one or more of the following:

- A fault handling interrupt.
- An error recovery interrupt.
- An in-band error response.

Each of these might generate an exception at a PE. If the PE reads the Error Record registers at the node, after taking an exception generated by such a signal from a node, then the read must return the updated values.

This is true for both error records accessed through the memory-mapped registers and error records accessed through the System registers. For memory-mapped registers, this assumes that the memory-mapped registers are mapped as a Device type that does not permit read speculation.

Direct reads of the System registers, including error record System registers, can occur speculatively and out-of-order relative to other instructions executed on the same PE.

Direct reads and writes of the error records through the ERX* System registers are indirect reads of ERRSELR_EL1 or ERRSELR.

3.3 Error recovery interrupt

If an error recovery interrupt is implemented by a node, then the set of controls for enabling error recovery interrupts is IMPLEMENTATION DEFINED:

- A control for enabling the error recovery interrupt on Deferred errors, ERR<n>CTLR.DUI, might be implemented:
  - If the DUI control is implemented, it enables the error recovery interrupt for Deferred errors.
  - If the DUI control is not implemented, the error recovery interrupt is never generated for Deferred errors.
- A control for enabling the error recovery interrupt on Uncorrected errors, ERR<n>CTLR.UI, might be implemented:
  - If the UI control is implemented, it enables the error recovery interrupt for Uncorrected errors.
  - If the UI control is not implemented, the error recovery interrupt is always enabled for Uncorrected errors.

If an error recovery interrupt is not implemented by a node, these controls are not implemented.

For each implemented control, it is further IMPLEMENTATION DEFINED whether there is a single control or separate controls for reads and writes.

3.4 Fault handling interrupt

If a fault handling interrupt is implemented by a node, then the set of controls for enabling fault handling interrupts is IMPLEMENTATION DEFINED:

- A control for generating the fault handling interrupt on Corrected errors, ERR<n>CTLR.CFI, might be implemented. If the CFI control is implemented:
  - The CFI control enables the fault handling interrupt for Corrected error events.
  - The ERR<n>CTLR.FI control must also be implemented and enables the fault handling interrupt for Deferred and Uncorrected errors.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- If the CFI control is not implemented, a control for generating the fault handling interrupt on all detected errors, ERR<n>CTLR.FI, might be implemented:
  - If the FI control is implemented, it enables the fault handling interrupt for all Corrected error events, Deferred and Uncorrected errors.
  - If the FI control is not implemented, the fault handling interrupt is always enabled for all Corrected error events, Deferred and Uncorrected errors.

If a fault handling interrupt is not implemented by a node, these controls are not implemented.

A *Corrected error event* is either:

- When a counter overflows and sets an overflow bit, if the node implements a Corrected error counter. See Corrected error counting.
- Each detected Corrected error, otherwise.

For each implemented control, it is further IMPLEMENTATION DEFINED whether there is a single control or separate controls for reads and writes.

The fault handling interrupt is generated when the node detects an error, even if the error syndrome is discarded because the error record already records a higher priority error.

3.5 In-band error signaling (external aborts)

If support for in-band error signaling, also referred to as external aborts, is implemented by a node, then a control for signaling this might be implemented. This control is ERR<n>CTLR.UE:

- If the control is implemented, it enables external abort signaling for all Uncorrected errors.
- If the control is not implemented, then external aborts are always enabled for all Uncorrected errors.

If external aborts are not implemented by a node, this control is not implemented.

For this control, it is further IMPLEMENTATION DEFINED whether there is a single control or separate ERR<n>CTLR.{RUE, WUE} controls for reads and writes.

When the node signals an Uncorrected error using external abort, it sets ERR<n>STATUS.ER to 1.

3.6 Standard format Corrected error counter

The architecture defines standard formats for a Corrected error counter (CE counter) that can be indicated in ERR<n>FR, and, if implemented, in ERR<n>MISC0.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and an overflow bit is set.

An implementation might include a pair of counters. In such an arrangement, the first (repeat) error counter counts errors at the same location. Errors in other locations are counted in a second (other) counter. The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set.

—— Note ——————————

IMPLEMENTATION DEFINED forms of counters, including other sizes, other overflow models, and other ERR<n>MISC0 or ERR<n>MISC1 locations, might be implemented.

————————————————

See Corrected error counting.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

3.7 Error recovery and fault handling signaling

Error recovery and fault handling interrupts are normally routed using the interrupt controller.

It is IMPLEMENTATION DEFINED whether each node uses independent interrupts for fault handling and error recovery, or whether it shares the fault handling interrupts or error recovery interrupts, or both, of other nodes.

It is IMPLEMENTATION DEFINED whether interrupts are edge-triggered or level-sensitive.

If the interrupt is level-sensitive, it is asserted by the node while it is enabled in ERR<n>CTLR and the appropriate error flag or flags in the corresponding ERR<n>STATUS register are set, or (if implemented and applicable) the overflow bit in the Corrected error counter is set, or both. That is:

- The fault handling interrupt is asserted while any of the following apply:
  - Fault handling interrupts on all detected errors are enabled, the ERR<n>STATUS.V bit is set to 1, and either or both of the ERR<n>STATUS.{DE,UE} bits are set to 1.
  - Fault handling interrupts on Corrected errors are enabled and either:
    - If the node implements a Corrected error counter, both the ERR<n>STATUS.V bit and the counter overflow bit are set to 1.
    - If the node does not implement a Corrected error counter, both the ERR<n>STATUS.V bit is set to 1 and the ERR<n>STATUS.CE field is nonzero.
- The error recovery interrupt is asserted while any of the following apply:
  - Error recovery interrupts on Uncorrected errors are enabled and both the ERR<n>STATUS.V and ERR<n>STATUS.UE bits are set to 1.
  - Error recovery interrupts on Deferred errors are enabled and both the ERR<n>STATUS.V and ERR<n>STATUS.DE bits are set to 1.

If the interrupt is edge-triggered, then an enabled interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The standard error record reserves a set of register locations for programming MSIs. In addition, a recommended layout for these registers is provided. See ERRIRQCR, ERR<irq>CR0, ERR<irq>CR1, ERR<irq>CR2, and ERRIRQSR.

When an error is detected and recorded, or an interrupt becomes enabled, the state of the interrupts must be updated in finite time.

3.8 Error Recovery reset

A system comprises multiple power and logical domains, each of which might implement one or more reset signals.

This specification defines two classes of reset:

- Cold reset is asserted to a component when it transitions from a powered off state to a powered on state. Cold reset initializes the component to a known initial state. No state is preserved from the previous powered off state.
- Error Recovery reset is an optional reset that might be applied at any other time. System Error Recovery reset initializes the component to a known state. Unlike Cold reset, any recorded error syndrome information is preserved over a System Error Recovery reset.

The way in which these resets map to other resets is IMPLEMENTATION DEFINED. The mechanisms for asserting resets are IMPLEMENTATION DEFINED.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

For a PE, the Error Recovery reset might be implemented by the Warm reset defined by the ARMv8-A architecture. If Warm reset is implemented, it must preserve the error records in the PE.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4 RAS Extension Registers

4.1 Memory-mapped view

This section defines the OPTIONAL registers that a memory-mapped component might implement in a system implementing the RAS Extension.

A memory-mapped component might implement several error records in a group, relating to one or more nodes. That is, a group comprises one or more nodes and each node has one or more error records. For each error record, the registers follow the same format as the System registers, only without the ERRSELR mechanism (the record is selected by address), and with the addition of a group status register.

The memory access sizes that are supported by the memory-mapped component are as described for other memory-mapped components in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*, except that it is IMPLEMENTATION DEFINED whether a word-aligned 32-bit access to either half of a doubleword-aligned 64-bit register is supported if there is no PE in the system that supports AArch32.

Error Record registers, including memory mapped view describes a group with up to 56 records, the most that can be contained in a 4KB group. Extra records might be added by increasing the page size. In this view, ERRDEVID indicates the highest numbered index of the error records that can be accessed.

4.2 Reset values

Unless otherwise stated the reset values of all registers are IMPLEMENTATION DEFINED and might be UNKNOWN. Unless explicitly stated, the reset value refers to both Error Recovery and Cold reset.

Where a Cold reset value is explicitly stated, the register is unchanged on an Error Recovery reset.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.3 Writes to ERR<*n*>STATUS

After reading an ERR<n>STATUS register, software must clear the valid bits in the register to allow new errors to be recorded. During this period a new error could have overwritten the syndrome for the previously read error. If the register were read/write, then this would be lost.

To prevent this, most bits use a modified version of write-one-to-clear:

- Writes to the {UE, DE, CE} bits are ignored if the OF bit is set and is not being cleared.
- Writes to the V bit are ignored if any of the {UE, DE, CE} bits are set and are not being cleared.
- Writes to the {AV, MV} valid bits and {ER, PN, UET, IERR, SERR} syndrome fields are ignored if the highest priority error status bit (in priority order: UE, DE, CE) is set and not being cleared.

The {AV, V, UE, ER, OF, MV, CE, DE, PN, UET} fields are write-one-to-clear, meaning writes of zero are ignored, and a write of one or all-ones to the field clears the field to zero. The {IERR, SERR} fields are read/write fields, although the set of permitted values that can be written to the fields is IMPLEMENTATION DEFINED. Some of these fields are also defined as UNKNOWN when certain combinations of the {V, DE, UE} status fields are zero. For more information, see ERR<n>STATUS.

These rules allow a node to implement a field as a fixed read-only value. To ensure correct and portable operation, software must write ones to nonzero write-one-to-clear fields, and zero to read/write fields when clearing the status fields to zero.

```
// ERRSTATUS[] (assignment form)
// =============================
// For a system register, n = UInt(ERRSELR_EL1.SEL)
ERRSTATUS[integer n] = bits(32) w
    STATUS = _ERRSTATUS[n];                                     // previous value (physical register)
    c = (STATUS<31:20> AND NOT(w<31:20>)):Zeros(4):w<15:0>;     // candidate value
    if c.OF == '1' then c.<UE,DE,CE> = STATUS.<UE,DE,CE>;       // do not clear UE/DE/CE if OF set
    if !IsZero(c.<UE,DE,CE>) then c.V = STATUS.V;               // do not clear V if UE/DE/CE set
    if (c.UE != '0' ||                                          // do not clear syndrome if not
        (STATUS.UE == '0' && c.DE != '0') ||                    // clearing highest priority error
        (STATUS.<UE,DE> == '00' && c.CE != '00')) then
        c.<AV,ER,MV,PN,UET,IERR,SERR> = STATUS.<AV,ER,MV,PN,UET,IERR,SERR>;
    _ERRSTATUS[n] = c;
```


Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4 AArch64 System registers

This section describes the System registers that are new for AArch64 as part of the RAS Extension.

For information on the System registers that are changed for AArch64 as part of the RAS Extension, see *The Reliability, Availability, and Serviceability (RAS) Extension* in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

4.4.1 Register index

Table 1: AArch64 System register map

op0	op1	CRn	CRm	op2	Access	Register	Description
3	0	c5	c3	0	RO	ERRIDR_EL1	Error Record ID Register
				1	R/W	ERRSELR_EL1	Error Record Select Register
			c4	0	RO	ERXFR_EL1	Selected Error Record Feature Register
				1	R/W	ERXCTLR_EL1	Selected Error Record Control Register
				2	R/W	ERXSTATUS_EL1	Selected Error Record Primary Status Register
				3	R/W	ERXADDR_EL1	Selected Error Record Address Register
			c5	0	R/W	ERXMISC0_EL1	Selected Error Record Miscellaneous Register 0
				1	R/W	ERXMISC1_EL1	Selected Error Record Miscellaneous Register 1
		c12	c1	1	R/W	DISR_EL1	Deferred Interrupt Status Register
	4	c5	c2	3	R/W	VSESR_EL2	Virtual SError Exception Syndrome Register
		c12	c1	1	R/W	VDISR_EL2	Virtual Deferred Interrupt Status Register

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.2 DISR_EL1, Deferred Interrupt Status Register

The DISR_EL1 characteristics are:

Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

Usage constraints

Direct reads and writes of DISR_EL1 are UNDEFINED at EL0.

If EL2 is implemented and HCR_EL2.AMO == 1, then direct reads and writes of DISR_EL1 at Non-secure EL1 access VDISR_EL2.

If not accessing VDISR_EL2, EL3 is implemented, the PE is in Non-debug state, and SCR_EL3.EA == 1, then DISR_EL1 is RAZ/WI at EL2 and EL1.

An indirect write to DISR_EL1 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Traps and enables

None.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of DISR_EL1 are UNDEFINED otherwise.

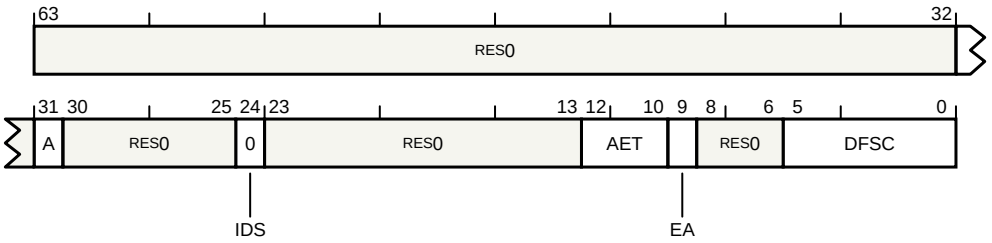
If the implementation supports AArch32 at EL1, DISR_EL1 is architecturally mapped to DISR.

Attributes

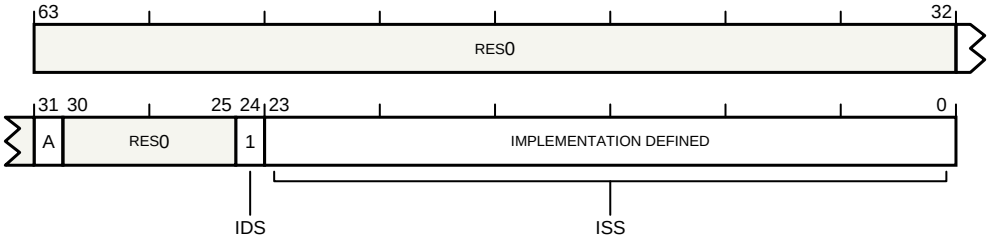
DISR_EL1 is a 64-bit read/write AArch64 System register accessed using MRS and MSR of S3_0_c12_c1_1.

Field descriptions

When IDS == 0, the DISR_EL1 bit assignments are:



When IDS == 1, the DISR_EL1 bit assignments are:



Bits [63:32,30:25]

Reserved. This field is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

IDS, bit [24]

Indicates whether the deferred SError interrupt was of an IMPLEMENTATION DEFINED type. The possible values of this bit are:

`0` Deferred error uses architecturally-defined format.

`1` Deferred error uses IMPLEMENTATION DEFINED format.

ISS, bits [23:0], when IDS == 1

IMPLEMENTATION DEFINED SError interrupt syndrome. See the description of ESR_EL*x*[23:0] for an SError interrupt.

Bits [23:13,8:6], when IDS == 0

Reserved. This field is RES0.

AET, bits [12:10], when IDS == 0

Asynchronous Error Type. See the description of ESR_EL*x*.AET for an SError interrupt.

EA, bit [9], when IDS == 0

External Abort Type. See the description of ESR_EL*x*.EA for an SError interrupt.

DFSC, bits [5:0], when IDS == 0

Fault Status Code. See the description of ESR_EL*x*.DFSC for an SError interrupt.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.3 ERRIDR_EL1, Error Record ID Register

The ERRIDR_EL1 characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Usage constraints

Direct reads of ERRIDR_EL1 are UNDEFINED at EL0.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads of ERRIDR_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads of ERRIDR_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads of ERRIDR_EL1 are UNDEFINED otherwise.

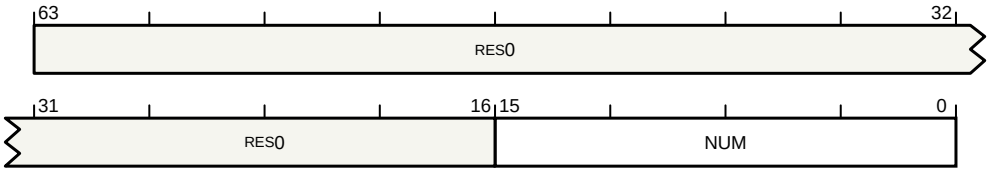
If the implementation supports AArch32 at EL1, ERRIDR_EL1 is architecturally mapped to ERRIDR.

Attributes

ERRIDR_EL1 is a 64-bit read-only AArch64 System register accessed using `MRS` of `S3_0_c5_c3_0`.

Field descriptions

The ERRIDR_EL1 bit assignments are:



Bits [63:16]

Reserved. This field is RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers, plus one. Zero indicates no records can be accessed through the Error Record System registers. Each implemented record is owned by a node. A node might own multiple records.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.4 ERRSELR_EL1, Error Record Select Register

The ERRSELR_EL1 characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Usage constraints

Direct reads and writes of ERRSELR_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1 indicates that zero records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR_EL1 is UNDEFINED or RES0.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERRSELR_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERRSELR_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERRSELR_EL1 are UNDEFINED otherwise.

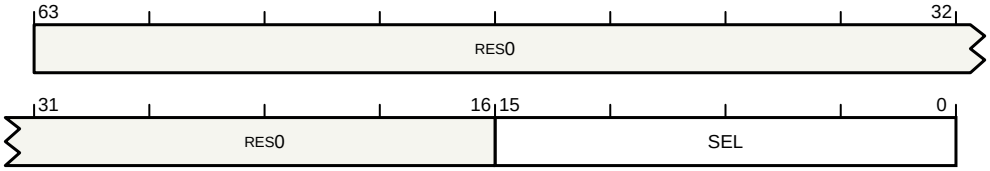
If the implementation supports AArch32 at EL1, ERRSELR_EL1 is architecturally mapped to ERRSELR.

Attributes

ERRSELR_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c3_1`.

Field descriptions

The ERRSELR_EL1 bit assignments are:



Bits [63:16]

Reserved. This field is RES0.

SEL, bits [15:0]

Selects the record accessed through the ERX registers.

For example, if ERRSELR_EL1.SEL is set to 4, then direct reads and writes of ERXSTATUS_EL1 access ERR4STATUS.

If ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then all of the following apply:

- The value read back from ERRSELR_EL1.SEL is UNKNOWN.
- One of the following occurs:
  - An UNKNOWN record is selected.
  - The ERX* registers are RAZ/WI.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

- ERX* register reads and writes are NOPs.
- ERX* register reads and writes are UNDEFINED.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.5 ERXADDR_EL1, Selected Error Record Address Register

The ERXADDR_EL1 characteristics are:

Purpose

Accesses ERR<n>ADDR for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads and writes of ERXADDR_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXADDR_EL1 is RAZ/WI.
- Direct reads and writes of ERXADDR_EL1 are NOPs.
- Direct reads and writes of ERXADDR_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERXADDR_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERXADDR_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERXADDR_EL1 are UNDEFINED otherwise.

Attributes

ERXADDR_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c4_3`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.6 ERXCTLR_EL1, Selected Error Record Control Register

The ERXCTLR_EL1 characteristics are:

Purpose

Accesses ERR<n>CTLR for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads and writes of ERXCTLR_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXCTLR_EL1 is RAZ/WI.
- Direct reads and writes of ERXCTLR_EL1 are NOPs.
- Direct reads and writes of ERXCTLR_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERXCTLR_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERXCTLR_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERXCTLR_EL1 are UNDEFINED otherwise.

Attributes

ERXCTLR_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c4_1`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.7 ERXFR_EL1, Selected Error Record Feature Register

The ERXFR_EL1 characteristics are:

Purpose

Accesses ERR<n>FR for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads of ERXFR_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXFR_EL1 is RAZ/WI.
- Direct reads of ERXFR_EL1 are NOPs.
- Direct reads of ERXFR_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads of ERXFR_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads of ERXFR_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads of ERXFR_EL1 are UNDEFINED otherwise.

Attributes

ERXFR_EL1 is a 64-bit read-only AArch64 System register accessed using `MRS` of `S3_0_c5_c4_0`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.8 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0

The ERXMISC0_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC0 for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads and writes of ERXMISC0_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC0_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC0_EL1 are NOPs.
- Direct reads and writes of ERXMISC0_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC0_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC0_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC0_EL1 are UNDEFINED otherwise.

Attributes

ERXMISC0_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c5_0`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.9 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1

The ERXMISC1_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC1 for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads and writes of ERXMISC1_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC1_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC1_EL1 are NOPs.
- Direct reads and writes of ERXMISC1_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC1_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC1_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC1_EL1 are UNDEFINED otherwise.

Attributes

ERXMISC1_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c5_1`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.10 ERXSTATUS_EL1, Selected Error Record Primary Status Register

The ERXSTATUS_EL1 characteristics are:

Purpose

Accesses ERR<n>STATUS for the error record selected by ERRSELR_EL1.SEL.

Usage constraints

Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED at EL0.

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXSTATUS_EL1 is RAZ/WI.
- Direct reads and writes of ERXSTATUS_EL1 are NOPs.
- Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and HCR_EL2.TERR == 1, then direct reads and writes of ERXSTATUS_EL1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL3 is implemented and SCR_EL3.TERR == 1, then direct reads and writes of ERXSTATUS_EL1 at EL2 and EL1 generate a Trap exception to EL3.

Configurations

Present only if the RAS Extension is implemented. Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED otherwise.

Attributes

ERXSTATUS_EL1 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_0_c5_c4_2`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.11 VDISR_EL2, Virtual Deferred Interrupt Status Register

The VDISR_EL2 characteristics are:

Purpose

Records that a virtual SError interrupt has been consumed by an ESB instruction executed at Non-secure EL1.

Usage constraints

Direct reads and writes of VDISR_EL2 are UNDEFINED at EL1 and EL0.

If EL1 is using AArch64 and HCR_EL2.AMO == 1, then direct reads and writes of DISR_EL1 at Non-secure EL1 access VDISR_EL2.

If EL1 is using AArch32 and HCR_EL2.AMO == 1, then direct reads and writes of DISR at Non-secure EL1 access VDISR_EL2.

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 or DISR occurring in program order after the ESB instruction.

Traps and enables

None.

Configurations

VDISR_EL2 is RES0 at EL3 if EL2 is not implemented.

Present only if the RAS Extension is implemented. Direct reads and writes of VDISR_EL2 are UNDEFINED otherwise.

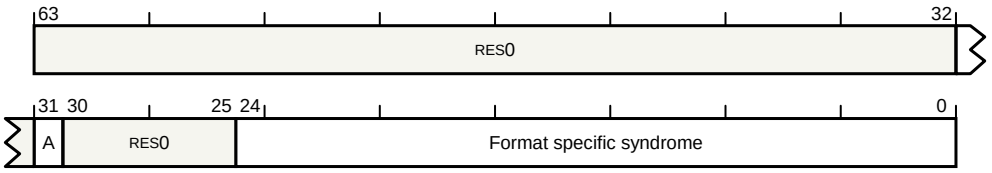
If the implementation supports AArch32 at EL2, VDISR_EL2 is architecturally mapped to VDISR.

Attributes

VDISR_EL2 is a 64-bit read/write AArch64 System register accessed using MRS and MSR of S3_4_c12_c1_1.

VDISR_EL2

The VDISR_EL2 bit assignments are:



Bits [63:32,30:25]

Reserved. This field is RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

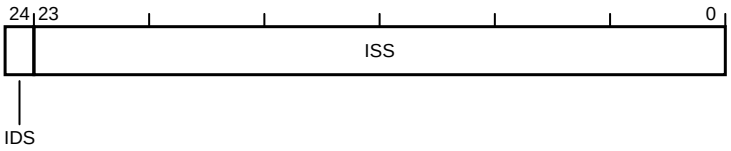
Format specific syndrome, bits [24:0]

The format of this field depends on the Exception level, Execution state, and translation table descriptor format used.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Format specific syndrome when EL1 is using AArch64

The format specific syndrome when EL1 is using AArch64 bit assignments are:



IDS, bit [24]

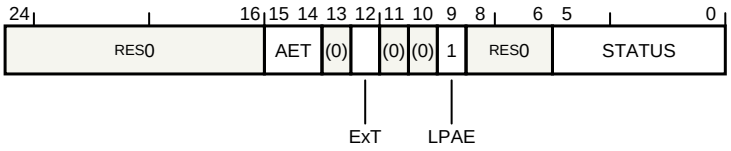
Contains the value from VSESR_EL2.IDS.

ISS, bits [23:0]

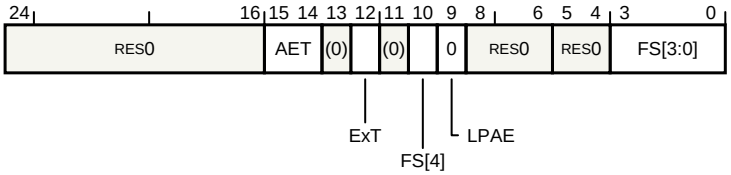
Contains the value from VSESR_EL2.ISS.

Format specific syndrome when EL1 is using AArch32

When EL1 is using the Long-descriptor format, the format specific syndrome when EL1 is using AArch32 bit assignments are:



When EL1 is using the Short-descriptor format, the format specific syndrome when EL1 is using AArch32 bit assignments are:



Bits [24:16,13,11,8:6]

Reserved. This field is RES0.

AET, bits [15:14]

Contains the value from VSESR_EL2.AET.

ExT, bit [12]

Contains the value from VSESR_EL2.ExT.

FS, bits [10,3:0], when EL1 is using the Short-descriptor format

Fault status code. Set to `0b10110` when an `ESB` instruction defers a virtual SError interrupt. The possible values of this field are:

`0b10110` Asynchronous SError interrupt.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Bit [10], when EL1 is using the Long-descriptor format

Reserved. This bit is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

LPAE, bit [9]

Format. The possible values of this bit are:

`1` Using the Long-descriptor translation table format.

`0` Using the Short-descriptor translation table format.

STATUS, bits [5:0], when EL1 is using the Long-descriptor format

Fault status code. Set to `0b010001` when an `ESB` instruction defers a virtual SError interrupt. The possible values of this field are:

`0b010001` Asynchronous SError interrupt.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Bits [5:4], when EL1 is using the Short-descriptor format

Reserved. This field is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.4.12 VSESR_EL2, Virtual SError Exception Syndrome Register

The VSESR_EL2 characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError interrupt exception:

- If the virtual SError interrupt is taken to EL1 using AArch64, then VSESR_EL2 provides the syndrome value reported in ESR_EL1.
- If the virtual SError interrupt is taken to EL1 using AArch32, then VSESR_EL2 provides the syndrome values that are reported in DFSR.{AET, ExT} and the remainder of the DFSR is set as defined by VMSAv8-32, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

Usage constraints

Direct reads and writes of VSESR_EL2 are UNDEFINED at EL1 and EL0.

Traps and enables

None.

Configurations

VSESR_EL2 is RES0 at EL3 if EL2 is not implemented.

Present only if the RAS Extension is implemented. Direct reads and writes of VSESR_EL2 are UNDEFINED otherwise.

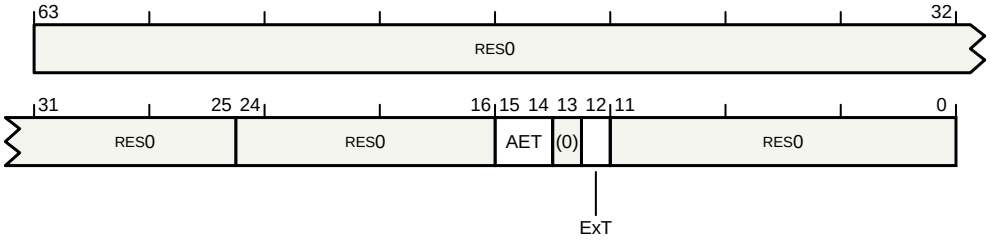
If the implementation supports AArch32 at EL2, VSESR_EL2 is architecturally mapped to VDFSR.

Attributes

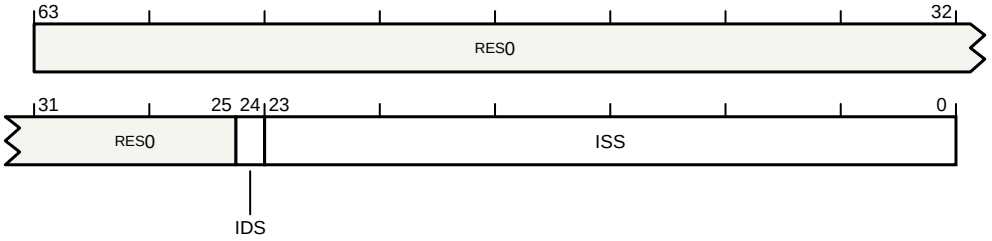
VSESR_EL2 is a 64-bit read/write AArch64 System register accessed using `MRS` and `MSR` of `S3_4_c5_c2_3`.

Field descriptions

When EL1 is using AArch32, the VSESR_EL2 bit assignments are:



When EL1 is using AArch64, the VSESR_EL2 bit assignments are:



Bits [63:25]

Reserved. This field is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

IDS, bit [24], when EL1 is using AArch64

On taking a virtual SError interrupt to EL1 using AArch64, ESR_EL1[24] is set to VSESR_EL2.IDS.

Bits [24:16,13,11:0], when EL1 is using AArch32

Reserved. This field is RES0.

ISS, bits [23:0], when EL1 is using AArch64

On taking a virtual SError interrupt to EL1 using AArch64, ESR_EL1[23:0] is set to VSESR_EL2.ISS.

AET, bits [15:14], when EL1 is using AArch32

On taking a virtual SError interrupt to EL1 using AArch32, DFSR[15:14] is set to VSESR_EL2.AET.

ExT, bit [12], when EL1 is using AArch32

On taking a virtual SError interrupt to EL1 using AArch32, DFSR[12] is set to VSESR_EL2.ExT.

4.5 AArch32 System registers

This section describes the System registers that are new for AArch32 as part of the RAS Extension.

For information on the System registers that are changed for AArch32 as part of the RAS Extension, see *The Reliability, Availability, and Serviceability (RAS) Extension* in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

4.5.1 Register index

Table 2: AArch32 System register (coproc=0b1111) map

opc1	CRn	CRm	opc2	Access	Register	Description
0	c5	c3	0	RO	ERRIDR	Error Record ID Register
			1	R/W	ERRSELR	Error Record Select Register
		c4	0	RO	ERXFR	Selected Error Record Feature Register
			1	R/W	ERXCTLR	Selected Error Record Control Register
			2	R/W	ERXSTATUS	Selected Error Record Primary Status Register
			3	R/W	ERXADDR	Selected Error Record Address Register
			4	RO	ERXFR2	Selected Error Record Feature Register 2
			5	R/W	ERXCTLR2	Selected Error Record Control Register 2
			7	R/W	ERXADDR2	Selected Error Record Address Register 2
		c5	0	R/W	ERXMISC0	Selected Error Record Miscellaneous Register 0
			1	R/W	ERXMISC1	Selected Error Record Miscellaneous Register 1
			4	R/W	ERXMISC2	Selected Error Record Miscellaneous Register 2
			5	R/W	ERXMISC3	Selected Error Record Miscellaneous Register 3
	c12	c1	1	R/W	DISR	Deferred Interrupt Status Register
4	c5	c2	3	R/W	VDFSR	Virtual SError Exception Syndrome Register
	c12	c1	1	R/W	VDISR	Virtual Deferred Interrupt Status Register

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.2 DISR, Deferred Interrupt Status Register

The DISR characteristics are:

Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

Usage constraints

Direct reads and writes of DISR are UNDEFINED at EL0.

If EL2 is implemented, then direct reads and writes of DISR at Non-secure EL1 access:

- VDISR_EL2 if EL2 is using AArch64 and HCR_EL2.AMO is set to 1.
- VDISR if EL2 is using AArch32 and HCR.AMO is set to 1.

If not accessing VDISR_EL2 or VDISR, then DISR is RAZ/WI at EL2 and EL1 if any of the following are true:

- EL3 is implemented and using AArch64, the PE is in Non-debug state, and SCR_EL3.EA == 1.
- EL3 is implemented and using AArch32, the PE is in Non-debug state, and SCR.EA == 1.

An indirect write to DISR made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR occurring in program order after the ESB.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, and HSTR_EL2.T12 == 1, then direct reads and writes of DISR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, and HSTR.T12 == 1, then direct reads and writes of DISR in Non-secure state at EL1 generate a Hyp Trap exception.

Configurations

There is one instance of DISR that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of DISR are UNDEFINED otherwise.

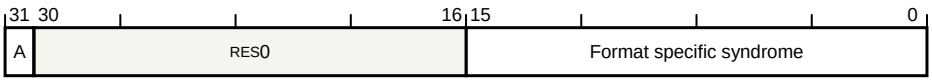
If the highest implemented Exception level is using AArch64, DISR is architecturally mapped to DISR_EL1.

Attributes

DISR is a 32-bit read/write AArch32 System register accessed using MRC and MCR of `p15,0,c12,c1,1`.

DISR

The DISR bit assignments are:



A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [30:16]

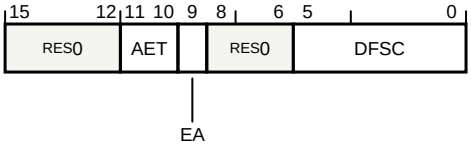
Reserved. This field is RES0.

Format specific syndrome, bits [15:0]

The format of this field depends on the Exception level and translation table descriptor format used.

Format specific syndrome for an ESB instruction executed at EL2

The format specific syndrome for an ESB instruction executed at EL2 bit assignments are:



Bits [15:12,8:6]

Reserved. This field is RES0.

AET, bits [11:10]

Asynchronous Error Type. See the description of HSR.AET for an SError interrupt.

EA, bit [9]

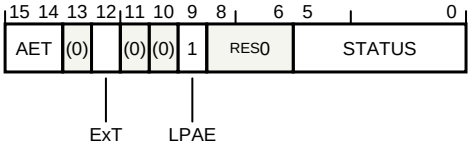
External Abort Type. See the description of HSR.EA for an SError interrupt.

DFSC, bits [5:0]

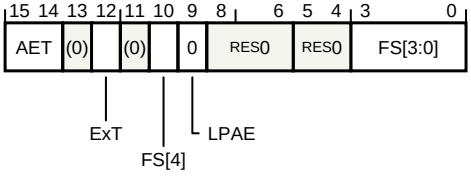
Fault Status Code. See the description of HSR.DFSC for an SError interrupt.

Format specific syndrome for an ESB instruction executed at EL0 or EL1

When EL1 is using the Long-descriptor format, the format specific syndrome for an ESB instruction executed at EL0 or EL1 bit assignments are:



When EL1 is using the Short-descriptor format, the format specific syndrome for an ESB instruction executed at EL0 or EL1 bit assignments are:



AET, bits [15:14]

Asynchronous Error Type. See the description of DFSR.AET for an SError interrupt.

Bits [13,11,8:6]

Reserved. This field is RES0.

ExT, bit [12]

External Abort Type. See the description of DFSR.ExT for an SError interrupt.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

FS, bits [10,3:0], when EL1 is using the Short-descriptor format

Fault Status Code. See the description of DFSR.FS for an SError interrupt.

Bit [10], when EL1 is using the Long-descriptor format

Reserved. This bit is RES0.

LPAE, bit [9]

Format. The possible values of this bit are:

`0` Using the Short-descriptor translation table format.

`1` Using the Long-descriptor translation table format.

STATUS, bits [5:0], when EL1 is using the Long-descriptor format

Fault Status Code. See the description of DFSR.DFSC for an SError interrupt.

Bits [5:4], when EL1 is using the Short-descriptor format

Reserved. This field is RES0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.3 ERRIDR, Error Record ID Register

The ERRIDR characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Usage constraints

Direct reads of ERRIDR are UNDEFINED at EL0.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads of ERRIDR in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads of ERRIDR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads of ERRIDR in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads of ERRIDR in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads of ERRIDR at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads of ERRIDR in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERRIDR that is used in both Secure and Non-secure states.

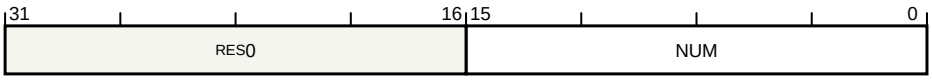
Present only if the RAS Extension is implemented. Direct reads of ERRIDR are UNDEFINED otherwise.

Attributes

ERRIDR is a 32-bit read-only AArch32 System register accessed using `MRC` of `p15,0,c5,c3,0`.

Field descriptions

The ERRIDR bit assignments are:



Bits [31:16]

Reserved. This field is RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers, plus one. Zero indicates that no records can be accessed through the Error Record System registers. Each implemented record is owned by a node. A node might own multiple records.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.4 ERRSELR, Error Record Select Register

The ERRSELR characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Usage constraints

Direct reads and writes of ERRSELR are UNDEFINED at EL0.

If ERRIDR indicates that zero records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR is UNDEFINED or RES0.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERRSELR in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERRSELR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERRSELR in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERRSELR in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERRSELR at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERRSELR in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERRSELR that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERRSELR are UNDEFINED otherwise.

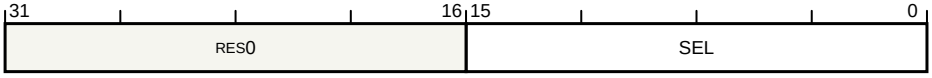
If the highest implemented Exception level is using AArch64, ERRSELR is architecturally mapped to ERRSELR_EL1.

Attributes

ERRSELR is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c3,1`.

Field descriptions

The ERRSELR bit assignments are:



Bits [31:16]

Reserved. This field is RES0.

SEL, bits [15:0]

Selects the record accessed through the ERX registers.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

For example, if ERRSELR.SEL is set to 4, then direct reads and writes of ERXSTATUS access ERR4STATUS.

If ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then all of the following apply:

- The value read back from ERRSELR.SEL is UNKNOWN.
- One of the following occurs:
  - An UNKNOWN record is selected.
  - The ERX* registers are RAZ/WI.
  - ERX* register reads and writes are NOPs.
  - ERX* register reads and writes are UNDEFINED.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.5 ERXADDR, Selected Error Record Address Register

The ERXADDR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>ADDR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXADDR are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXADDR is RAZ/WI.
- Direct reads and writes of ERXADDR are NOPs.
- Direct reads and writes of ERXADDR are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXADDR in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXADDR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXADDR in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXADDR in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXADDR at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXADDR in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXADDR that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXADDR are UNDEFINED otherwise.

Attributes

ERXADDR is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c4,3`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.6 ERXADDR2, Selected Error Record Address Register 2

The ERXADDR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>ADDR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXADDR2 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXADDR2 is RAZ/WI.
- Direct reads and writes of ERXADDR2 are NOPs.
- Direct reads and writes of ERXADDR2 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXADDR2 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXADDR2 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXADDR2 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXADDR2 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXADDR2 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXADDR2 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXADDR2 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXADDR2 are UNDEFINED otherwise.

Attributes

ERXADDR2 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c4,7`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.7 ERXCTLR, Selected Error Record Control Register

The ERXCTLR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>CTLR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXCTLR are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXCTLR is RAZ/WI.
- Direct reads and writes of ERXCTLR are NOPs.
- Direct reads and writes of ERXCTLR are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXCTLR in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXCTLR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXCTLR in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXCTLR in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXCTLR at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXCTLR in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXCTLR that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXCTLR are UNDEFINED otherwise.

Attributes

ERXCTLR is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c4,1`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.8 ERXCTLR2, Selected Error Record Control Register 2

The ERXCTLR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>CTLR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXCTLR2 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXCTLR2 is RAZ/WI.
- Direct reads and writes of ERXCTLR2 are NOPs.
- Direct reads and writes of ERXCTLR2 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXCTLR2 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXCTLR2 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXCTLR2 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXCTLR2 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXCTLR2 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXCTLR2 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXCTLR2 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXCTLR2 are UNDEFINED otherwise.

Attributes

ERXCTLR2 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c4,5`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.9 ERXFR, Selected Error Record Feature Register

The ERXFR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>FR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads of ERXFR are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXFR is RAZ/WI.
- Direct reads of ERXFR are NOPs.
- Direct reads of ERXFR are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads of ERXFR in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads of ERXFR in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads of ERXFR in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads of ERXFR in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads of ERXFR at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads of ERXFR in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXFR that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads of ERXFR are UNDEFINED otherwise.

Attributes

ERXFR is a 32-bit read-only AArch32 System register accessed using `MRC` of `p15,0,c5,c4,0`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.10 ERXFR2, Selected Error Record Feature Register 2

The ERXFR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>FR for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads of ERXFR2 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXFR2 is RAZ/WI.
- Direct reads of ERXFR2 are NOPs.
- Direct reads of ERXFR2 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads of ERXFR2 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads of ERXFR2 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads of ERXFR2 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads of ERXFR2 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads of ERXFR2 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads of ERXFR2 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXFR2 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads of ERXFR2 are UNDEFINED otherwise.

Attributes

ERXFR2 is a 32-bit read-only AArch32 System register accessed using `MRC` of `p15,0,c5,c4,4`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.11 ERXMISC0, Selected Error Record Miscellaneous Register 0

The ERXMISC0 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC0 for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXMISC0 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC0 is RAZ/WI.
- Direct reads and writes of ERXMISC0 are NOPs.
- Direct reads and writes of ERXMISC0 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC0 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXMISC0 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXMISC0 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXMISC0 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC0 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXMISC0 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXMISC0 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC0 are UNDEFINED otherwise.

Attributes

ERXMISC0 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c5,0`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.12 ERXMISC1, Selected Error Record Miscellaneous Register 1

The ERXMISC1 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC0 for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXMISC1 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC1 is RAZ/WI.
- Direct reads and writes of ERXMISC1 are NOPs.
- Direct reads and writes of ERXMISC1 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC1 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXMISC1 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXMISC1 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXMISC1 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC1 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXMISC1 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXMISC1 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC1 are UNDEFINED otherwise.

Attributes

ERXMISC1 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c5,1`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.13 ERXMISC2, Selected Error Record Miscellaneous Register 2

The ERXMISC2 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC1 for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXMISC2 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC2 is RAZ/WI.
- Direct reads and writes of ERXMISC2 are NOPs.
- Direct reads and writes of ERXMISC2 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC2 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXMISC2 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXMISC2 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXMISC2 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC2 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXMISC2 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXMISC2 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC2 are UNDEFINED otherwise.

Attributes

ERXMISC2 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c5,4`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.14 ERXMISC3, Selected Error Record Miscellaneous Register 3

The ERXMISC3 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC1 for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXMISC3 are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC3 is RAZ/WI.
- Direct reads and writes of ERXMISC3 are NOPs.
- Direct reads and writes of ERXMISC3 are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXMISC3 in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXMISC3 in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXMISC3 in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXMISC3 in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXMISC3 at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXMISC3 in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXMISC3 that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXMISC3 are UNDEFINED otherwise.

Attributes

ERXMISC3 is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c5,5`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.15 ERXSTATUS, Selected Error Record Primary Status Register

The ERXSTATUS characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>STATUS for the error record selected by ERRSELR.SEL.

Usage constraints

Direct reads and writes of ERXSTATUS are UNDEFINED at EL0.

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXSTATUS is RAZ/WI.
- Direct reads and writes of ERXSTATUS are NOPs.
- Direct reads and writes of ERXSTATUS are UNDEFINED.

Traps and enables

Subject to the exception prioritization rules:

- If EL2 is implemented and using AArch64, then:
  - If HCR_EL2.TERR == 1, then direct reads and writes of ERXSTATUS in Non-secure state at EL1 generate a Trap exception to EL2.
  - If HSTR_EL2.T5 == 1, then direct reads and writes of ERXSTATUS in Non-secure state at EL1 generate a Trap exception to EL2.
- If EL2 is implemented and using AArch32, then:
  - If HCR2.TERR == 1, then direct reads and writes of ERXSTATUS in Non-secure state at EL1 generate a Hyp Trap exception.
  - If HSTR.T5 == 1, then direct reads and writes of ERXSTATUS in Non-secure state at EL1 generate a Hyp Trap exception.
- If EL3 is implemented and using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXSTATUS at EL2 and EL1 generate a Trap exception to EL3.
- If EL3 is implemented and using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXSTATUS in modes other than Monitor mode generate a Monitor Trap exception.

Configurations

There is one instance of ERXSTATUS that is used in both Secure and Non-secure states.

Present only if the RAS Extension is implemented. Direct reads and writes of ERXSTATUS are UNDEFINED otherwise.

Attributes

ERXSTATUS is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,0,c5,c4,2`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.16 VDFSR, Virtual SError Exception Syndrome Register

The VDFSR characteristics are:

Purpose

Provides the syndrome value reported to software in DFSR.{AET, ExT} on taking a virtual SError interrupt exception. The remainder of the DFSR is set as defined by VMSAv8-32, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

Usage constraints

Direct reads and writes of VDFSR are UNDEFINED at EL1 and EL0 and, if EL3 is implemented and using AArch32, in:

- All Secure privileged modes other than Monitor mode.
- Monitor mode when SCR.NS == 0.

Traps and enables

None.

Configurations

There is one instance of VDFSR that is used in both Secure and Non-secure states.

Present only if all of the following apply:

- EL2 is implemented and using AArch32.
- The RAS Extension is implemented.

Direct reads and writes of VDFSR are UNDEFINED otherwise.

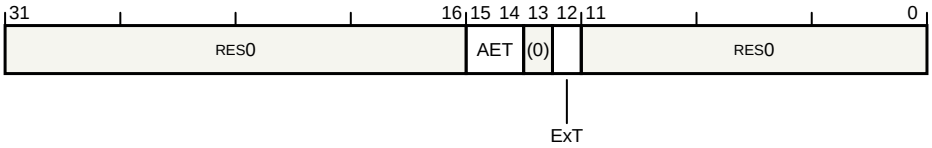
If the highest implemented Exception level is using AArch64, VDFSR is architecturally mapped to VSESR_EL2.

Attributes

VDFSR is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,4,c5,c2,3`.

Field descriptions

The VDFSR bit assignments are:



Bits [31:16,13,11:0]

Reserved. This field is RES0.

AET, bits [15:14]

On taking a virtual SError interrupt to EL1 using AArch32 because HCR.VA == 1, DFSR[15:14] is set to VDFSR.AET.

ExT, bit [12]

On taking a virtual SError interrupt to EL1 using AArch32 because HCR.VA == 1, DFSR[12] is set to VDFSR.ExT.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.5.17 VDISR, Virtual Deferred Interrupt Status Register

The VDISR characteristics are:

Purpose

Records that a virtual SError interrupt has been consumed by an ESB instruction executed at Non-secure EL1.

Usage constraints

Direct reads and writes of VDISR are UNDEFINED at EL1 and EL0 and, if EL3 is implemented and using AArch32, in:

- All Secure privileged modes other than Monitor mode.
- Monitor mode when SCR.NS == 0.

If HCR.AMO == 1, then direct reads and writes of DISR at Non-secure EL1 access VDISR.

An indirect write to VDISR made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR occurring in program order after the ESB.

Traps and enables

None.

Configurations

There is one instance of VDISR that is used in both Secure and Non-secure states.

Present only if all of the following apply:

- EL2 is implemented and using AArch32.
- The RAS Extension is implemented.

Direct reads and writes of VDISR are UNDEFINED otherwise.

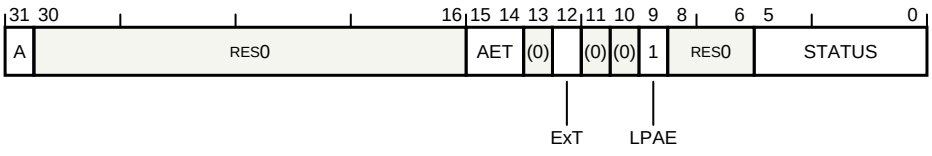
If the highest implemented Exception level is using AArch64, VDISR is architecturally mapped to VDISR_EL2.

Attributes

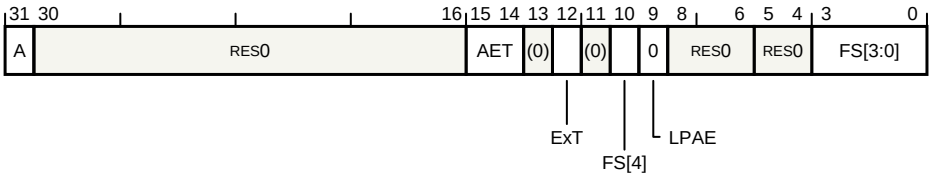
VDISR is a 32-bit read/write AArch32 System register accessed using `MRC` and `MCR` of `p15,4,c12,c1,1`.

Field descriptions

When EL1 is using the Long-descriptor format, the VDISR bit assignments are:



When EL1 is using the Short-descriptor format, the VDISR bit assignments are:



Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

Bits [30:16,13,11,8:6]

Reserved. This field is RES0.

AET, bits [15:14]

Contains the value from VDFSR.AET.

ExT, bit [12]

Contains the value from VDFSR.ExT.

FS, bits [10,3:0], when EL1 is using the Short-descriptor format

Fault status code. Set to `0b10110` when an ESB instruction defers a virtual SError interrupt. The possible values of this field are:

`0b10110` Asynchronous SError interrupt.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Bit [10], when EL1 is using the Long-descriptor format

Reserved. This bit is RES0.

LPAE, bit [9]

Format. The possible values of this bit are:

`1` Using the Long-descriptor translation table format.

`0` Using the Short-descriptor translation table format.

STATUS, bits [5:0], when EL1 is using the Long-descriptor format

Fault status code. Set to `0b010001` when an ESB instruction defers a virtual SError interrupt. The possible values of this field are:

`0b010001` Asynchronous SError interrupt.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Bits [5:4], when EL1 is using the Short-descriptor format

Reserved. This field is RES0.

4.6 Error Record registers, including memory mapped view

This section describes the Error Record registers that can be accessed either:

- Through the indirection mechanism described in Error record System register view.
- As memory-mapped registers, as described in Memory-mapped view.

4.6.1 Register index

Using AArch32 System registers

Table 3: Using AArch32 System registers, System register map

Use	To Access	Access	Description
ERXADDR	ERR<n>ADDR[31:0]	R/W	Error Record Address Register

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Use	To Access	Access	Description
ERXADDR2	ERR<n>ADDR[63:32]	R/W	Error Record Address Register
ERXCTLR	ERR<n>CTLR[31:0]	R/W	Error Record Control Register
ERXCTLR2	ERR<n>CTLR[63:32]	R/W	Error Record Control Register
ERXFR	ERR<n>FR[31:0]	RO	Error Record Feature Register
ERXFR2	ERR<n>FR[63:32]	RO	Error Record Feature Register
ERXMISC0	ERR<n>MISC0[31:0]	R/W	Error Record Miscellaneous Register 0
ERXMISC1	ERR<n>MISC0[63:32]	R/W	Error Record Miscellaneous Register 0
ERXMISC2	ERR<n>MISC1[31:0]	R/W	Error Record Miscellaneous Register 1
ERXMISC3	ERR<n>MISC1[63:32]	R/W	Error Record Miscellaneous Register 1
ERXSTATUS	ERR<n>STATUS[31:0]	R/W	Error Record Primary Status Register

Using AArch64 System registers

Table 4: Using AArch64 System registers, System register map

Use	To Access	Access	Description
ERXADDR_EL1	ERR<n>ADDR	R/W	Error Record Address Register
ERXCTLR_EL1	ERR<n>CTLR	R/W	Error Record Control Register
ERXFR_EL1	ERR<n>FR	RO	Error Record Feature Register
ERXMISC0_EL1	ERR<n>MISC0	R/W	Error Record Miscellaneous Register 0
ERXMISC1_EL1	ERR<n>MISC1	R/W	Error Record Miscellaneous Register 1
ERXSTATUS_EL1	ERR<n>STATUS	R/W	Error Record Primary Status Register

Memory-mapped register map

Table 5: Memory-mapped register map

Offset	Access	Size	Register	Description
0x000+64×*n*	RO	64	ERR<n>FR	Error Record Feature Register
0x008+64×*n*	R/W	64	ERR<n>CTLR	Error Record Control Register
0x010+64×*n*	R/W	64	ERR<n>STATUS	Error Record Primary Status Register
0x018+64×*n*	R/W	64	ERR<n>ADDR	Error Record Address Register
0x020+64×*n*	R/W	64	ERR<n>MISC0	Error Record Miscellaneous Register 0
0x028+64×*n*	R/W	64	ERR<n>MISC1	Error Record Miscellaneous Register 1
0xE00+8×*n*	RO	64	ERRGSR<*n*>	Error Group Status Register
0xE80	R/W	64	ERRFHICR0	ERR<irq>CR0, Error Interrupt Configuration Register 0
0xE80+8×*n*	R/W	64	ERRIRQCR<*n*>	Generic Error Interrupt Configuration Register
0xE88	R/W	32	ERRFHICR1	ERR<irq>CR1, Error Interrupt Configuration Register 1
0xE8C	R/W	32	ERRFHICR2	ERR<irq>CR2, Error Interrupt Configuration Register 2
0xE90	R/W	64	ERRERICR0	ERR<irq>CR0, Error Interrupt Configuration Register 0

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Offset	Access	Size	Register	Description
0xE98	R/W	32	ERRERICR1	ERR<irq>CR1, Error Interrupt Configuration Register 1
0xE9C	R/W	32	ERRERICR2	ERR<irq>CR2, Error Interrupt Configuration Register 2
0xEF8	R/W	64	ERRIRQSR	Error Interrupt Status Register
0xFA8	RO	64	ERRDEVAFF	Device Affinity Register
0xFBC	RO	32	ERRDEVARCH	Device Architecture Register
0xFC8	RO	32	ERRDEVID	Error Record Device ID Register
0xFD0	RO	32	ERRPIDR4	Peripheral Identification Register 4
0xFD4	RAZ	32	ERRPIDR5	Peripheral Identification Register 5
0xFD8	RAZ	32	ERRPIDR6	Peripheral Identification Register 6
0xFDC	RAZ	32	ERRPIDR7	Peripheral Identification Register 7
0xFE0	RO	32	ERRPIDR0	Peripheral Identification Register 0
0xFE4	RO	32	ERRPIDR1	Peripheral Identification Register 1
0xFE8	RO	32	ERRPIDR2	Peripheral Identification Register 2
0xFEC	RO	32	ERRPIDR3	Peripheral Identification Register 3
0xFF0	RO	32	ERRCIDR0	Component Identification Register 0
0xFF4	RO	32	ERRCIDR1	Component Identification Register 1
0xFF8	RO	32	ERRCIDR2	Component Identification Register 2
0xFFC	RO	32	ERRCIDR3	Component Identification Register 3

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.2 ERR<*irq*>CR0, Error Interrupt Configuration Register 0

The ERR<*irq*>CR0 characteristics are:

Purpose

Recommended interrupt configuration register.

Usage constraints

None.

Configurations

Present only if the register locations for programming MSIs use the recommended layout. RES0 otherwise.

ERR<*irq*>CR0 is architecturally mapped to ERRIRQCR<n×2>.

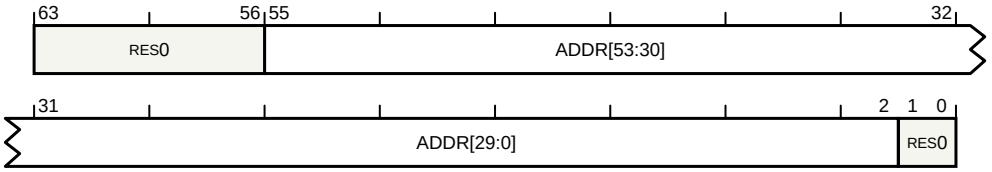
Attributes

ERR<*irq*>CR0 are 64-bit read/write memory-mapped registers located at:

- Offset `0xE90` for ERRERICR0.
- Offset `0xE80` for ERRFHICR0.

Field descriptions

The ERR<*irq*>CR0 bit assignments are:



Bits [63:56,1:0]

Reserved. This field is RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. Specifies the address that the node writes to when signaling an interrupt.

The size of a physical address is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RAZ/WI.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.3 ERR<*irq*>CR1, Error Interrupt Configuration Register 1

The ERR<*irq*>CR1 characteristics are:

Purpose

Recommended interrupt configuration register.

Usage constraints

None.

Configurations

Present only if the register locations for programming MSIs use the recommended layout. RES0 otherwise.

ERR<*irq*>CR1 is architecturally mapped to ERRIRQCR<n×2+1>[31:0].

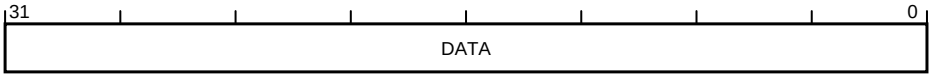
Attributes

ERR<*irq*>CR1 are 32-bit read/write memory-mapped registers located at:

- Offset `0xE98` for ERRERICR1.
- Offset `0xE88` for ERRFHICR1.

Field descriptions

The ERR<*irq*>CR1 bit assignments are:



DATA, bits [31:0]

Payload for a message signaled interrupt.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.4 ERR<*irq*>CR2, Error Interrupt Configuration Register 2

The ERR<*irq*>CR2 characteristics are:

Purpose

Recommended interrupt configuration register.

Usage constraints

None.

Configurations

Present only if the register locations for programming MSIs use the recommended layout. RES0 otherwise.

ERR<*irq*>CR2 is architecturally mapped to ERRIRQCR<n×2+1>[63:32].

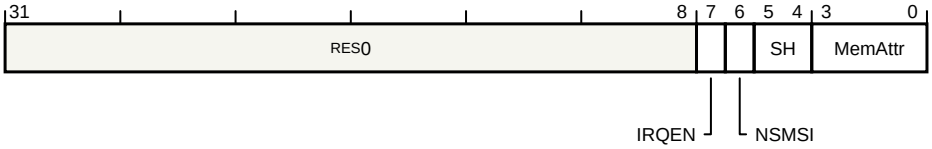
Attributes

ERR<*irq*>CR2 are 32-bit read/write memory-mapped registers located at:

- Offset `0xE9C` for ERRERICR2.
- Offset `0xE8C` for ERRFHICR2.

Field descriptions

The ERR<*irq*>CR2 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

IRQEN, bit [7]

Message signaled interrupt enable. Enables generation of message signaled interrupts. The possible values of this bit are:

`0` Disabled.

`1` Enabled.

This bit is RES0 if the component does not support disabling message signaled interrupts meaning message signaled interrupts are always enabled.

NSMSI, bit [6]

Security attribute. Defines the physical address space for message signaled interrupts. The possible values of this bit are:

`0` Secure.

`1` Non-secure.

If the component prohibits Non-secure writes and does not support configuring the Security attribute, then the Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

If the component allows Non-secure writes, then the Security attribute used for message signaled interrupts is Non-secure.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

This bit is RES0 if any of the following are true:

- The component allows Non-secure writes.
- The component does not support configuring the Security attribute.

SH, bits [5:4]

Shareability. Defines the Shareability domain for message signaled interrupts. The possible values of this field are:

`0b00` Not shared.

`0b10` Outer Shareable.

`0b11` Inner Shareable.

This field is RES0 if the component does not support configuring the Shareability domain, meaning the Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

Memory type. Defines the memory type for message signaled interrupts. The possible values of this field are:

`0b0000` Device-nGnRnE.

`0b0001` Device-nGnRE.

`0b0010` Device-nGRE.

`0b0011` Device-GRE.

`0b0101` Outer Non-cacheable, Inner Non-cacheable.

`0b0110` Outer Non-cacheable, Inner Write-Through Cacheable.

`0b0111` Outer Non-cacheable, Inner Write-Back Cacheable.

`0b1001` Outer Write-Through Cacheable, Inner Non-cacheable.

`0b1010` Outer Write-Through Cacheable, Inner Write-Through Cacheable.

`0b1011` Outer Write-Through Cacheable, Inner Write-Back Cacheable.

`0b1101` Outer Write-Back Cacheable, Inner Non-cacheable.

`0b1110` Outer Write-Back Cacheable, Inner Write-Through Cacheable.

`0b1111` Outer Write-Back Cacheable, Inner Write-Back Cacheable.

This field is RES0 if the component does not support configuring the memory type, meaning the memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

—— Note ——————————

This is the same format as the VMSAv8-64 stage 2 memory region attributes. See the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

————————————————

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.5 ERR<*n*>ADDR, Error Record Address Register

The ERR<*n*>ADDR characteristics are:

Purpose

If an error has an associated address, then this must be written to the address register when the error is recorded. It is IMPLEMENTATION DEFINED how the recorded addresses map to the software-visible physical addresses. Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

Usage constraints

Ignores writes if ERR<n>STATUS.AV is set to 1.

Configurations

Always implemented.

Attributes

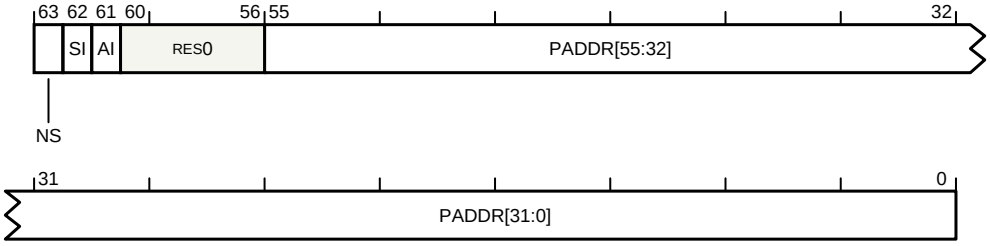
When accessed using System registers, ERR<*n*>ADDR is a 64-bit read/write register accessed using:

- `MRC` and `MCR` of ERXADDR for ERR<*n*>ADDR[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXADDR2 for ERR<*n*>ADDR[63:32] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXADDR_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>ADDR is a 64-bit read/write register located at offset `0x018` + 64×*n*.

Field descriptions

The ERR<*n*>ADDR bit assignments are:



NS, bit [63]

Non-secure attribute. The possible values of this bit are:

`0` The address is Secure.

`1` The address is Non-secure.

SI, bit [62]

Secure Incorrect. Indicates whether the NS bit is valid. The possible values of this bit are:

`0` The NS bit is correct. That is, it matches the programmers' view of the Non-secure attribute for this recorded location.

`1` The NS bit might not be correct, and might not match the programmers' view of the Non-secure attribute for the recorded location.

It is IMPLEMENTATION DEFINED whether this bit is read-only or read/write.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

AI, bit [61]

Address Incorrect. Indicates whether the PADDR field is a valid physical address that is known to match the programmers' view of the physical address for the recorded location. The possible values of this bit are:

`0` The PADDR field is a valid physical address. That is, it matches the programmers' view of the physical address for the recorded location.

`1` The PADDR field might not be a valid physical address, and might not match the programmers' view of the physical address for the recorded location.

It is IMPLEMENTATION DEFINED whether this bit is read-only or read/write.

Bits [60:56]

Reserved. This field is RES0.

PADDR, bits [55:0]

Physical Address. Address of the recorded location. If the physical address size implemented by this component is smaller than the size of this field, then high-order bits are unimplemented and either RES0 or have a fixed read-only IMPLEMENTATION DEFINED value. Low-order address bits might also be unimplemented and RES0, for example, if the physical address is always aligned to the size of a protection granule.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.6 ERR<*n*>CTLR, Error Record Control Register

The ERR<*n*>CTLR characteristics are:

Purpose

The error control register contains enable bits for the node that writes to this record:

- Enabling error detection and correction.
- Enabling an error recovery interrupt.
- Enabling a fault handling interrupt.
- Enabling error recovery reporting as a read or write error response.

For each bit, if the selected node does not support the feature, then the bit is RES0. The definition of each record is IMPLEMENTATION DEFINED.

Usage constraints

None.

Configurations

If a single node has multiple error records, then only the first error record has an ERR<*n*>CTLR register. For other records of the same node, ERR<*n*>CTLR is RES0.

Attributes

When accessed using System registers, ERR<*n*>CTLR is a 64-bit read/write register accessed using:

- `MRC` and `MCR` of ERXCTLR for ERR<*n*>CTLR[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXCTLR2 for ERR<*n*>CTLR[63:32] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXCTLR_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>CTLR is a 64-bit read/write register located at offset `0x008` + 64×*n*.

Preface

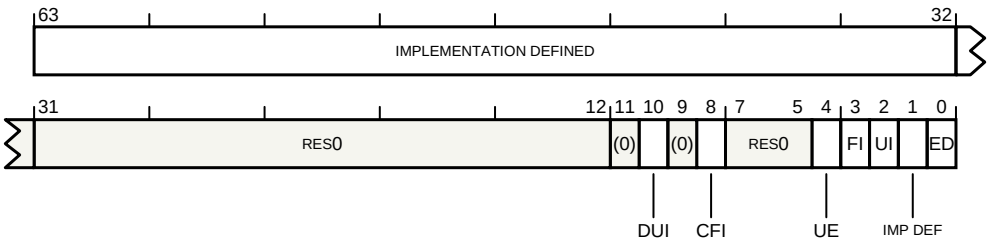
For some controls, it is IMPLEMENTATION DEFINED whether ERR<*n*>CTLR contains a single combined read/write control or separate read and write controls. This register description shows two possible combinations:

- All controls are combined.
- All controls are separate.

However, this is IMPLEMENTATION DEFINED for each control.

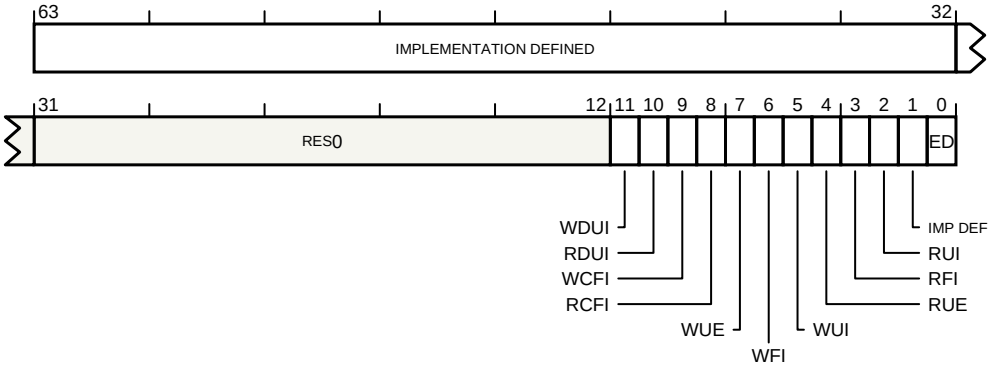
Field descriptions

When combined read/write control, the ERR<*n*>CTLR bit assignments are:



Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

When separate read/write controls, the ERR<*n*>CTLR bit assignments are:



Bits [63:32]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Bits [31:12]

Reserved. This field is RES0.

WDUI, bit [11], when separate read/write controls

Error recovery interrupt for deferred errors on writes enable. The definition is the same as the DUI bit, except it applies to writes only.

Bits [11,9,7:5], when combined read/write control

Reserved. This field is RES0.

DUI, bit [10], when combined read/write control

Error recovery interrupt for deferred errors enable. When enabled the error recovery interrupt is generated for all detected Deferred errors. The possible values of this bit are:

`0` Error recovery interrupt not generated for deferred errors.

`1` Error recovery interrupt generated for deferred errors.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

This bit is RES0 if the node does not support this control.

Note

Applies to both reads and writes.

RDUI, bit [10], when separate read/write controls

Error recovery interrupt for deferred errors on reads enable. The definition is the same as the DUI bit, except it applies to reads only.

WCFI, bit [9], when separate read/write controls

Fault handling interrupt for corrected errors on writes enable. The definition is the same as the CFI bit, except it applies to writes only.

CFI, bit [8], when combined read/write control

Fault handling interrupt for corrected errors enable.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

When enabled:

- If the node implements a Corrected error counter, then the fault handling interrupt is generated when the counter overflows and the overflow bit is set. See ERR<n>MISC0.
- Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

0 Fault handling interrupt not generated for corrected errors.

1 Fault handling interrupt generated for corrected errors.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

This bit is RES0 if the node does not support this control.

Note

Applies to both reads and writes.

RCFI, bit [8], when separate read/write controls

Fault handling interrupt for corrected errors on reads enable. The definition is the same as the CFI bit, except it applies to reads only.

WUE, bit [7], when separate read/write controls

In-band Uncorrected error reporting on writes enable. The definition is the same as the UE bit, except it applies to writes only.

WFI, bit [6], when separate read/write controls

Fault handling interrupt on writes enable. The definition is the same as the FI bit, except it applies to writes only.

WUI, bit [5], when separate read/write controls

Error recovery interrupt on writes enable. The definition is the same as the UI bit, except it applies to writes only.

UE, bit [4], when combined read/write control

In-band Uncorrected error reporting enable. When enabled, responses to transactions that detect an uncorrected error that cannot be deferred are signaled as a detected error (external abort). The possible values of this bit are:

0 External abort response for uncorrected errors disabled.

1 External abort response for uncorrected errors enabled.

This bit is RES0 if the node does not support this control.

Note

Applies to both reads and writes.

RUE, bit [4], when separate read/write controls

In-band Uncorrected error reporting on reads enable. The definition is the same as the UE bit, except it applies to reads only.

FI, bit [3], when combined read/write control

Fault handling interrupt enable.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

When enabled:

- The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors.
- If the fault handling interrupt for corrected errors control is not implemented:
  - If the node implements a Corrected error counter, then the fault handling interrupt is also generated when the counter overflows and the overflow bit is set.
  - Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

`0` Fault handling interrupt disabled.

`1` Fault handling interrupt enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

This bit is RES0 if the node does not support this control.

—— Note ——————————

Applies to both reads and writes.

————————————————

RFI, bit [3], when separate read/write controls

Fault handling interrupt on reads enable. The definition is the same as the FI bit, except it applies to reads only.

UI, bit [2], when combined read/write control

Uncorrected error recovery interrupt enable. When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred. The possible values of this bit are:

`0` Error recovery interrupt disabled.

`1` Error recovery interrupt enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

This bit is RES0 if the node does not support this control.

—— Note ——————————

Applies to both reads and writes.

————————————————

RUI, bit [2], when separate read/write controls

Error recovery interrupt on reads enable. The definition is the same as the UI bit, except it applies to reads only.

Bit [1]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This bit reads as an IMPLEMENTATION DEFINED value and writes to this bit have IMPLEMENTATION DEFINED behavior.

ED, bit [0]

Error reporting and logging enable. When disabled, the node behaves as if error detection and correction are disabled, and no errors are recorded or signaled by the node. ARM recommends that,

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection. The possible values of this bit are:

`0` Error reporting disabled.

`1` Error reporting enabled.

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting is disabled. That is, even with error reporting disabled, the node might continue to silently correct errors. Uncorrectable errors might result in corrupt data being silently propagated by the node.

This bit is RES0 if the node does not support this control.

This bit resets to an IMPLEMENTATION DEFINED value on a Cold reset.

Note

If this node requires initialization after Cold reset to prevent signaling false errors, then ARM recommends this bit is set to 0 on Cold reset, meaning errors are not reported from Cold reset. This allows boot software to initialize a node without signaling errors. Software can enable error reporting after the node is initialized. Otherwise, the Cold reset value is IMPLEMENTATION DEFINED.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.7 ERR<*n*>FR, Error Record Feature Register

The ERR<*n*>FR characteristics are:

Purpose

Defines which of the common architecturally-defined features are implemented and, of the implemented features, which are software programmable.

Usage constraints

None.

Configurations

If error record *n* is not implemented, then ERR<*n*>FR is RES0.

If a single node has multiple error records, then only the first error record has an ERR<*n*>FR register. For other records of the same node, ERR<*n*>FR is RES0.

Attributes

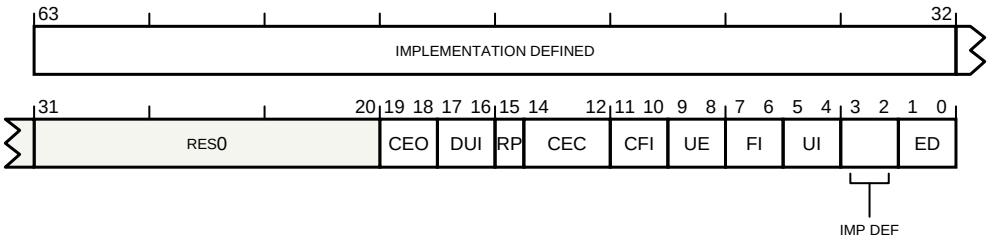
When accessed using System registers, ERR<*n*>FR is a 64-bit read-only register accessed using:

- `MRC` of ERXFR for ERR<*n*>FR[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` of ERXFR2 for ERR<*n*>FR[63:32] when ERRSELR.SEL is set to *n*.
- `MRS` of ERXFR_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>FR is a 64-bit read-only register located at offset `0x000` + 64×*n*.

Field descriptions

The ERR<*n*>FR bit assignments are:



Bits [63:32]

Reserved for identifying IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value.

Bits [31:20]

Reserved. This field is RES0.

CEO, bits [19:18]

Corrected Error overwrite. Indicates the behavior when a second Corrected error is detected after a first Corrected error has been recorded by the node. The defined values of this field are:

`0b00` Count Corrected error if a counter is implemented. Keep the previous error syndrome. If the counter overflows, or no counter is implemented, then ERR<n>STATUS.OF is set to 1.

`0b01` Count Corrected error. If ERR<n>STATUS.OF == 1 before the Corrected error is counted, then keep the previous syndrome. Otherwise the previous syndrome is overwritten. If the counter overflows, then ERR<n>STATUS.OF is set to 1.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field reads-as-zeros if no Corrected error counter is implemented. See also Writing the error record.

DUI, bits [17:16]

Error recovery interrupt for deferred errors. Indicates whether the node implements a control for enabling error recovery interrupts on deferred errors. The defined values of this field are:

`0b00` Does not support feature. ERR<n>CTLR.DUI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.DUI.

`0b11` Feature is controllable using ERR<n>CTLR.WDUI for writes and ERR<n>CTLR.RDUI for reads.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field is RES0 if ERR<*n*>FR.UI == `0b00`.

RP, bit [15]

Repeat counter. Indicates whether the node implements a repeat Corrected error counter. The defined values of this bit are:

`0` A single CE counter is implemented.

`1` A first (repeat) counter and a second (other) counter are implemented. The repeat counter is the same size as the primary error counter.

This bit is RES0 if ERR<*n*>FR.CEC == `0b00`.

CEC, bits [14:12]

Corrected Error Counter. Indicates whether the node implements a standard Corrected error counter (CE counter) mechanism in ERR<n>MISC0. The defined values of this field are:

`0b000` Does not implement the standard Corrected error counter model.

`0b010` Implements an 8-bit Corrected error counter in ERR<n>MISC0[39:32].

`0b100` Implements a 16-bit Corrected error counter in ERR<n>MISC0[47:32].

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Note

Implementations might include other error counter models, or might include the standard model and not indicate this in ERR<*n*>FR.

CFI, bits [11:10]

Fault handling interrupt for corrected errors. Indicates whether the node implements a control for enabling fault handling interrupts on corrected errors. The defined values of this field are:

`0b00` Does not support feature, ERR<n>CTLR.CFI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.CFI.

`0b11` Feature is controllable using ERR<n>CTLR.WCFI for writes and ERR<n>CTLR.RCFI for reads.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

This field is RES0 if ERR<*n*>FR.FI == `0b00`.

UE, bits [9:8]

In-band uncorrected error reporting. Indicates whether the node implements in-band uncorrected error reporting (external aborts), and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`0b00` Does not support feature, ERR<n>CTLR.UE is RES0.

`0b01` Feature always enabled, ERR<n>CTLR.UE is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.UE.

`0b11` Feature is controllable using ERR<n>CTLR.WUE for writes and ERR<n>CTLR.RUE for reads.

FI, bits [7:6]

Fault handling interrupt. Indicates whether the node implements a fault handling interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`0b00` Does not support feature, ERR<n>CTLR.FI is RES0.

`0b01` Feature always enabled, ERR<n>CTLR.FI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.FI.

`0b11` Feature is controllable using ERR<n>CTLR.WFI for writes and ERR<n>CTLR.RFI for reads.

UI, bits [5:4]

Error recovery interrupt for uncorrected errors. Indicates whether the node implements an error recovery interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`0b00` Does not support feature, ERR<n>CTLR.UI is RES0.

`0b01` Feature always enabled, ERR<n>CTLR.UI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.UI.

`0b11` Feature is controllable using ERR<n>CTLR.WUI for writes and ERR<n>CTLR.RUI for reads.

Bits [3:2]

This field reads as an IMPLEMENTATION DEFINED value.

ED, bits [1:0]

Error reporting and logging. Indicates whether the node implements controls for enabling and disabling error reporting and logging. The defined values of this field are:

`0b01` Feature always enabled, ERR<n>CTLR.ED is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.ED.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.8 ERR<*n*>MISC0, Error Record Miscellaneous Register 0

The ERR<*n*>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers contain:

- Corrected error counter or counters, if the node supports the counting of Corrected errors.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

If the node supports the architecturally-defined error counter, then it is implemented in ERR<*n*>MISC0.

Usage constraints

If ERR<n>STATUS.MV == 1, then it is IMPLEMENTATION DEFINED whether fields of ERR<*n*>MISC0 ignore writes. ARM recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes. This prevents information being lost if an error is detected whilst the previous error is being logged.

Configurations

Always implemented.

Attributes

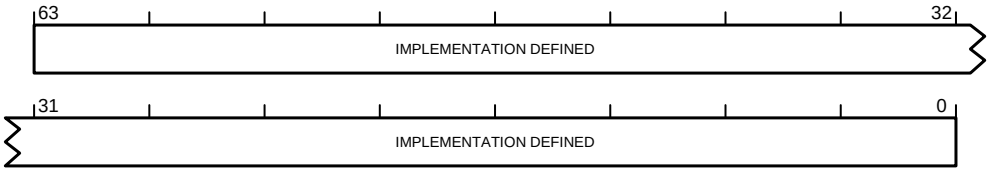
When accessed using System registers, ERR<*n*>MISC0 is a 64-bit read/write register accessed using:

- `MRC` and `MCR` of ERXMISC0 for ERR<*n*>MISC0[31:0] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXMISC0_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC1 for ERR<*n*>MISC0[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC0 is a 64-bit read/write register located at offset `0x020` + 64×*n*.

ERR<*n*>MISC0 (contents are IMPLEMENTATION DEFINED)

The ERR<*n*>MISC0 (contents are IMPLEMENTATION DEFINED) bit assignments are:



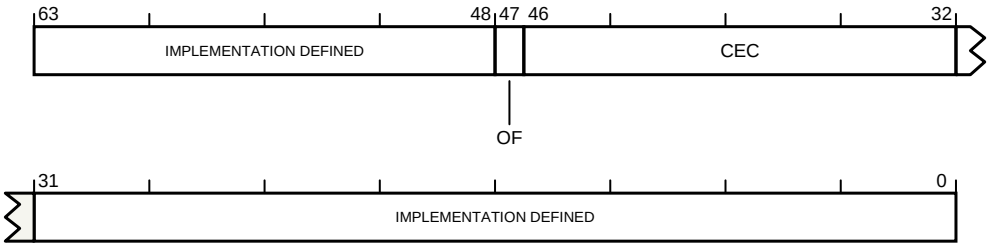
Bits [63:0], when contents are IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

ERR<*n*>MISC0 (standard 16-bit CE counter)

The ERR<*n*>MISC0 (standard 16-bit CE counter) bit assignments are:



Bits [63:48,31:0], when standard 16-bit CE counter

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OF, bit [47], when standard 16-bit CE counter

Sticky overflow bit. Set to 1 when the Corrected error count field is incremented and wraps through zero. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and the overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

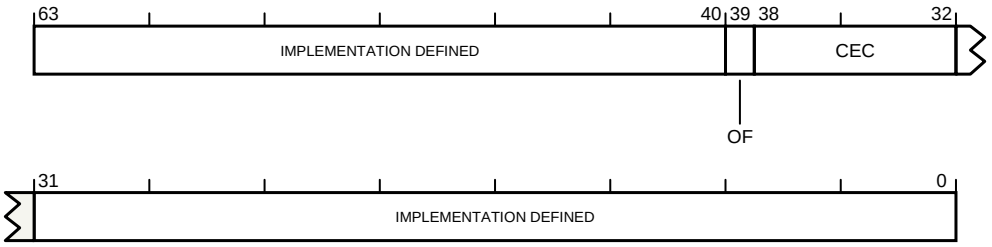
CEC, bits [46:32], when standard 16-bit CE counter

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

This field resets to an IMPLEMENTATION DEFINED value which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

ERR<*n*>MISC0 (standard 8-bit CE counter)

The ERR<*n*>MISC0 (standard 8-bit CE counter) bit assignments are:



Bits [63:40,31:0], when standard 8-bit CE counter

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

OF, bit [39], when standard 8-bit CE counter

Sticky overflow bit. Set to 1 when the Corrected error count field is incremented and wraps through zero. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and the overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

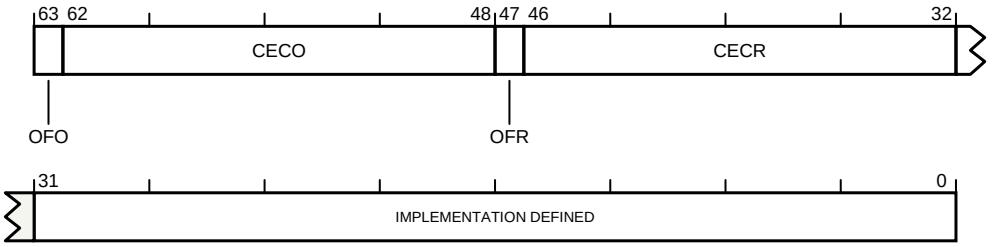
CEC, bits [38:32], when standard 8-bit CE counter

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

This field resets to an IMPLEMENTATION DEFINED value which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

ERR<*n*>MISC0 (standard 16-bit CE counter pair)

The ERR<*n*>MISC0 (standard 16-bit CE counter pair) bit assignments are:



OFO, bit [63], when standard 16-bit CE counter pair

Sticky overflow bit, other. Set to 1 when the Corrected error count, other, field is incremented and wraps through zero. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

CECO, bits [62:48], when standard 16-bit CE counter pair

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing CECR.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

OFR, bit [47], when standard 16-bit CE counter pair

Sticky overflow bit, repeat. Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

CECR, bits [46:32], when standard 16-bit CE counter pair

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

Note

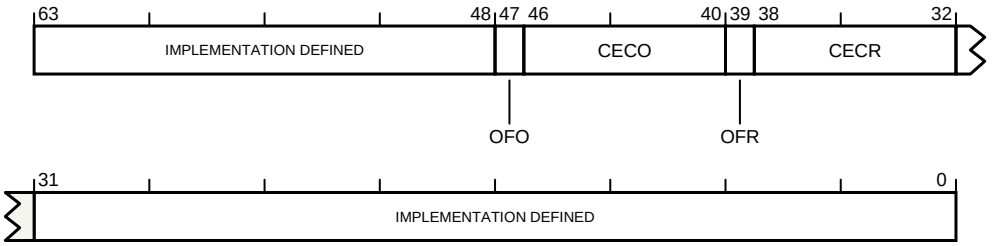
For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERRMISC<*n*> fields on a first Corrected error. CECR is then incremented for each subsequent Corrected Error in the same set and way.

Bits [31:0], when standard 16-bit CE counter pair

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

ERR<*n*>MISC0 (standard 8-bit CE counter pair)

The ERR<*n*>MISC0 (standard 8-bit CE counter pair) bit assignments are:



Bits [63:48,31:0], when standard 8-bit CE counter pair

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OFO, bit [47], when standard 8-bit CE counter pair

Sticky overflow bit, other. Set to 1 when the Corrected error count, other, field is incremented and wraps through zero. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

CECO, bits [46:40], when standard 8-bit CE counter pair

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing CECR.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

OFR, bit [39], when standard 8-bit CE counter pair

Sticky overflow bit, repeat. Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

CECR, bits [38:32], when standard 8-bit CE counter pair

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERRMISC<*n*> fields on a first Corrected error. CECR is then incremented for each subsequent Corrected Error in the same set and way.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.9 ERR<*n*>MISC1, Error Record Miscellaneous Register 1

The ERR<*n*>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers contain:

- Corrected error counter or counters, if the node supports the counting of Corrected errors.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

Usage constraints

If ERR<n>STATUS.MV == 1, then it is IMPLEMENTATION DEFINED whether fields of ERR<*n*>MISC1 ignore writes. ARM recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes. This prevents information being lost if an error is detected whilst the previous error is being logged.

Configurations

Always implemented.

Attributes

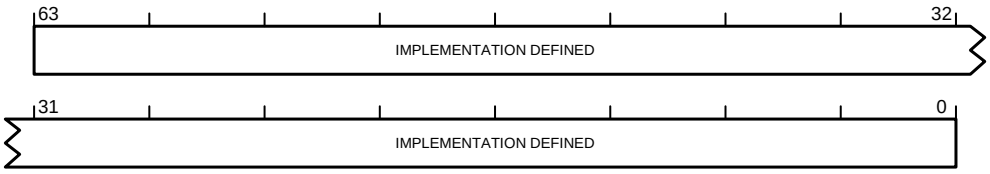
When accessed using System registers, ERR<*n*>MISC1 is a 64-bit read/write register accessed using:

- `MRS` and `MSR` of ERXMISC1_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC2 for ERR<*n*>MISC1[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC3 for ERR<*n*>MISC1[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC1 is a 64-bit read/write register located at offset `0x028` + 64×*n*.

Field descriptions

The ERR<*n*>MISC1 bit assignments are:



Bits [63:0]

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.10 ERR<*n*>STATUS, Error Record Primary Status Register

The ERR<*n*>STATUS characteristics are:

Purpose

Contains status information for the error record, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether an error record has been discarded because additional errors have been detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record registers contain valid information.
- Whether the error was recorded because poison data was detected or because a corrupt value was detected by an error detection code.
- A primary error code.
- An IMPLEMENTATION DEFINED extended error code.

Within this register:

- The {AV, V, MV} bits are valid bits that define whether the error record registers are valid.
- The {UE, OF, CE, DE, UET} bits encode the type of error or errors recorded.
- The {ER, PN, IERR, SERR} fields are syndrome fields.

Usage constraints

After reading the status register, software must clear the valid bits to allow new errors to be recorded. Between reading the register and clearing the valid bits, a new error might have overwritten the register. To prevent this error being lost, some control bits use a form of read/write-one-to-clear. For more information, see Writes to ERR<n>STATUS.

Configurations

Always implemented.

Attributes

When accessed using System registers, ERR<*n*>STATUS is a 64-bit read/write register accessed using:

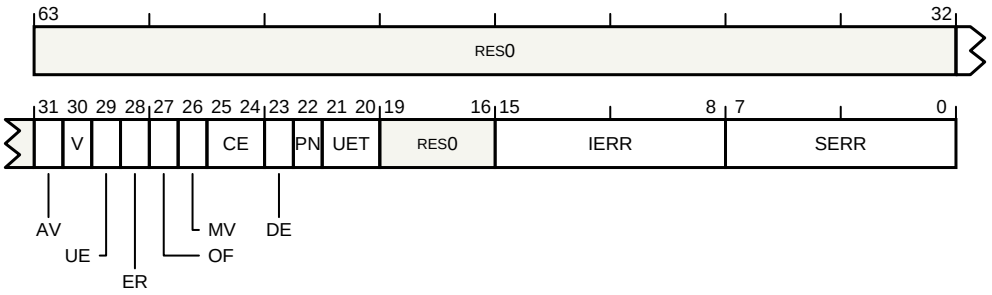
- `MRC` and `MCR` of ERXSTATUS for ERR<*n*>STATUS[31:0] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXSTATUS_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>STATUS is a 64-bit read/write register located at offset `0x010` + 64×*n*.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Field descriptions

The ERR<*n*>STATUS bit assignments are:



Bits [63:32,19:16]

Reserved. This field is RES0.

AV, bit [31]

Address Valid. The possible values of this bit are:

0 ERR<n>ADDR not valid.

1 ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

V, bit [30]

Status Register valid. The possible values of this bit are:

0 ERR<*n*>STATUS not valid.

1 ERR<*n*>STATUS valid. At least one error has been recorded.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

UE, bit [29]

Uncorrected error. The possible values of this bit are:

0 No errors have been detected, or all detected errors have been either corrected or deferred.

1 At least one detected error was not corrected and not deferred.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write. This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

ER, bit [28]

Error Reported. The possible values of this bit are:

0 No in-band error (external abort) reported.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

`1` An external abort was signaled by the node to the master making the access or other transaction. This can be because any of the following are true:

- The applicable one of the ERR<n>CTLR.{WUE,RUE,UE} bits is implemented and was set to 1 when an Uncorrected error was detected.
- The applicable one of the ERR<n>CTLR.{WUE,RUE,UE} bits is not implemented and the node always reports errors.

It is IMPLEMENTATION DEFINED whether this bit can be set to 1 by a Deferred error.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing ERR<*n*>STATUS.UE to 0, if this bit is never set to 1 by a Deferred error.
- Clearing both ERR<*n*>STATUS.{UE, DE} to 0, if this bit can be set to 1 by a Deferred error.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.UE is set to 0 and this bit is never set to 1 by a Deferred error.
- ERR<*n*>STATUS.{UE, DE} are both set to 0 and this bit can be set to 1 by a Deferred error.

This bit is read/write-one-to-clear.

Note

An external abort signaled by the node might be masked and not generate any exception.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This bit is set to 1 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 1.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and ERR<*n*>STATUS.CE != `0b00`. ERR<*n*>STATUS.CE might be updated for the new Corrected error.
- A Corrected error counter is implemented, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is set to 1 when one of the following occurs:

- A Deferred error is detected and ERR<*n*>STATUS.UE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, and either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 1.
- A Corrected error counter is implemented, either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 1, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is cleared to 0 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 0.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 0.
- A Corrected error is detected, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0 and ERR<*n*>STATUS.CE == `0b00`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

The IMPLEMENTATION DEFINED clearing of this bit might also depend on the value of the other error status bits.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.
- A direct write to this bit that clears this bit to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

The possible values of this bit are:

`0` If ERR<*n*>STATUS.UE == 1, then no error syndrome for an Uncorrected error has been discarded.

If ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 1, then no error syndrome for a Deferred error has been discarded.

If ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and a Corrected error counter is implemented, then the counter has not overflowed.

If ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, ERR<*n*>STATUS.CE != `0b00`, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded.

Note

This bit might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.

`1` At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

For more information, see Writing the error record.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

MV, bit [26]

Miscellaneous Registers Valid. The possible values of this bit are:

`0` ERR<n>MISC0 and ERR<n>MISC1 not valid.

`1` The IMPLEMENTATION DEFINED contents of the ERR<n>MISC0 and ERR<n>MISC1 registers contains additional information for an error recorded by this record.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

Note

If the ERR<n>MISC0 and ERR<n>MISC1 registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recently error that was not discarded.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

CE, bits [25:24]

Corrected error. The possible values of this field are:

`0b00` No errors were corrected.

`0b01` At least one transient error was corrected.

`0b10` At least one error was corrected.

`0b11` At least one persistent error was corrected.

The mechanism by which a node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to `0b10` when an error is corrected.

If this field is nonzero, then software must write ones to this field, to clear this field to zero, when clearing ERR<*n*>STATUS.V to 0.

This field ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write. This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

DE, bit [23]

Deferred error. The possible values of this bit are:

`0` No errors were deferred.

`1` At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write. This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

PN, bit [22]

Poison. The possible values of this bit are:

`0` Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC).

—— **Note** ——

If a producer node detects a corrupt value and defers the error by *producing* a poison value, then this bit is set to 0 at the producer node.

———————————

`1` Uncorrected error or Deferred error recorded because a poison value was detected.

—— **Note** ——

This might only be an indication of poison, because, in some EDC schemes, a poison value is encoded as an unlikely form of corrupt data, meaning it is possible to mistake a corrupt value as a poison value.

———————————

It is IMPLEMENTATION DEFINED whether a node can distinguish a poison value from a corrupt value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing both ERR<*n*>STATUS.{DE, UE} to 0.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.{DE, UE} are both set to 0.

This bit is read/write-one-to-clear.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error. The possible values of this field are:

`0b00` Uncorrected error, Uncontainable error (UC).

`0b01` Uncorrected error, Unrecoverable error (UEU).

`0b10` Uncorrected error, Latent or Restartable error (UEO).

`0b11` Uncorrected error, Signaled or Recoverable error (UER).

If this field is nonzero, then software must write ones to this field, to clear this field to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing ERR<*n*>STATUS.UE to 0.

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This field is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.UE is set to 0.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

—— Note ——————————

Software might use the information in the error record registers to determine what recovery is necessary.

————————————————

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code SERR value. Further IMPLEMENTATION DEFINED information can be placed in the MISC registers.

The subset of architecturally-defined values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write. This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Note

One or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

SERR, bits [7:0]

Architecturally-defined primary error code. Indicates the type of error. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry. The possible values of this field are:

0 No error.

1 IMPLEMENTATION DEFINED error.

2 Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.

3 IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.

4 Assertion failure. For example, consistency failure.

5 Error detected on internal data path. For example, parity on ALU result.

6 Data value from associative memory. For example, ECC error on cache data.

7 Address/control value from associative memory. For example, ECC error on cache tag.

8 Data value from a TLB. For example, ECC error on TLB data.

9 Address/control value from a TLB. For example, ECC error on TLB tag.

10 Data value from producer. For example, parity error on write data bus.

11 Address/control value from producer. For example, parity error on address bus.

12 Data value from (non-associative) external memory. For example, ECC error in SDRAM.

13 Illegal address (software fault). For example, access to unpopulated memory.

14 Illegal access (software fault). For example, byte write to word register.

15 Illegal state (software fault). For example, device not ready.

16 Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, and SIMD&FP registers are data registers.

17 Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, and SIMD&FP registers are control registers.

18 Error response from slave. For example, error response from cache write-back.

19 External timeout. For example, timeout on interaction with another node.

20 Internal timeout. For example, timeout on interface within the node.

21 Deferred error from slave not supported at master. For example, poisoned data received from a slave by a master that cannot defer the error further.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

The subset of architecturally-defined values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write. This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Note

One or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.11 ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

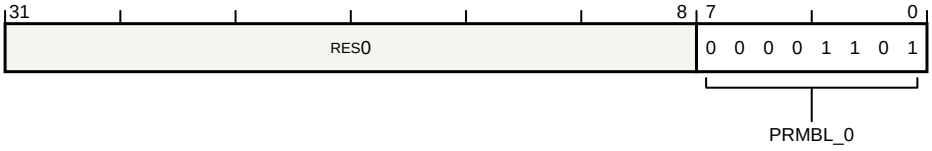
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFF0`.

Field descriptions

The ERRCIDR0 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_0, bits [7:0]

Component identification preamble. This field reads as `0x0D`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.12 ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

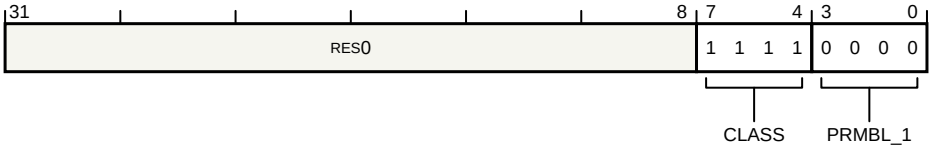
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFF4`.

Field descriptions

The ERRCIDR1 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

CLASS, bits [7:4]

Component class. The defined values of this field are:

`0xF` Generic peripheral.

This field reads as `0xF`.

PRMBL_1, bits [3:0]

Component identification preamble. This field reads as `0x0`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.13 ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

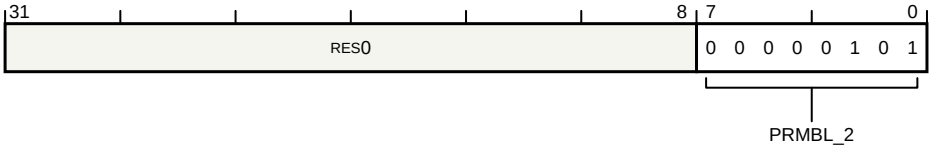
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFF8`.

Field descriptions

The ERRCIDR2 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_2, bits [7:0]

Component identification preamble. This field reads as `0x05`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.14 ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

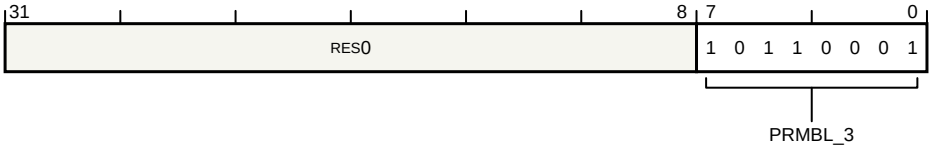
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFFC`.

Field descriptions

The ERRCIDR3 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_3, bits [7:0]

Component identification preamble. This field reads as `0xB1`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.15 ERRDEVAFF, Device Affinity Register

The ERRDEVAFF characteristics are:

Purpose

For a group that has affinity with a PE or cluster of PEs, ERRDEVAFF is a 64-bit read-only copy of MPIDR_EL1 or part of MPIDR_EL1.

Usage constraints

None.

Configurations

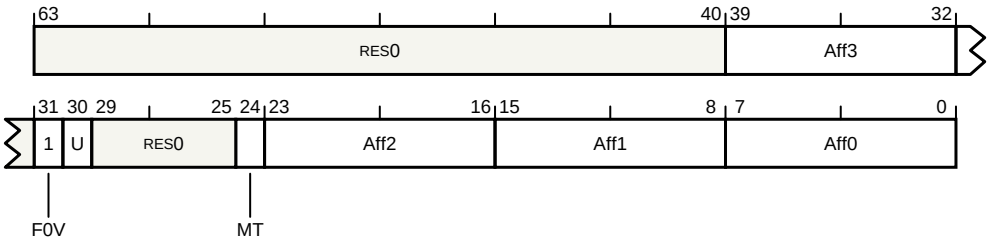
Present only if a group has affinity with a PE or cluster of PEs. RES0 otherwise.

Attributes

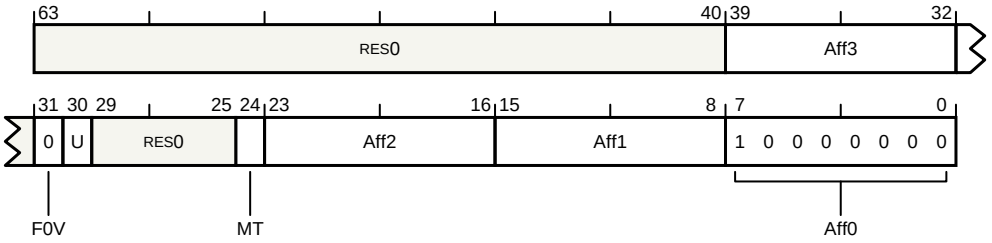
ERRDEVAFF is a 64-bit read-only memory-mapped register located at offset `0xFA8`.

Field descriptions

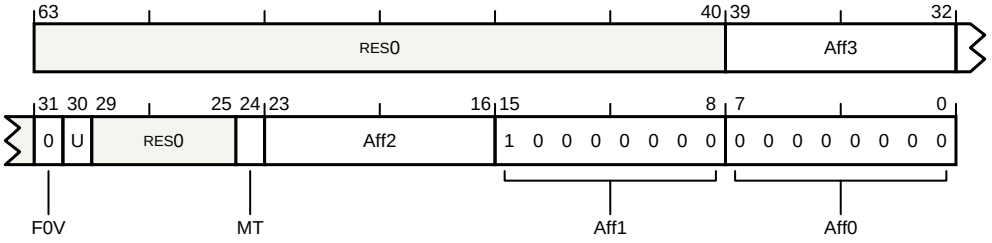
When level 0, the ERRDEVAFF bit assignments are:



When level 1, the ERRDEVAFF bit assignments are:

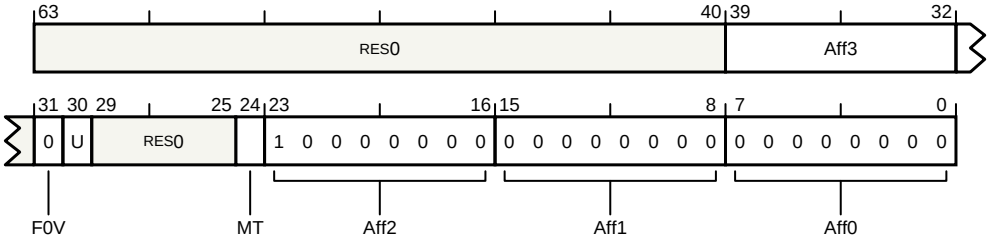


When level 2, the ERRDEVAFF bit assignments are:



Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

When level 3, the ERRDEVAFF bit assignments are:



Bits [63:40,29:25]

Reserved. This field is RES0.

Aff3, bits [39:32]

Affinity level 3. The MPIDR_EL1.Aff3 field viewed from the highest Exception level of the associated PE or PEs.

F0V, bit [31]

Indicates the Aff0 field is valid. The defined values of this bit are:

`0` Aff0 is not valid.

`1` Aff0 is valid.

U, bit [30]

Uniprocessor. The defined values of this bit are:

`0` PE is part of a multiprocessor system.

`1` PE is part of a uniprocessor system.

This bit is not valid and reads UNKNOWN if ERRDEVAFF.Aff0 is not valid.

MT, bit [24]

Multithreaded. The defined values of this bit are:

`0` Performance of PEs at the lowest affinity level is largely independent.

`1` Performance of PEs at the lowest affinity level is very interdependent.

This bit is not valid and reads UNKNOWN if ERRDEVAFF.Aff0 is not valid.

Aff2, bits [23:16], when level 0, when level 1, or when level 2

Affinity level 2. The MPIDR_EL1.Aff2 field viewed from the highest Exception level of the associated PE or PEs.

Aff2, bits [23:16], when level 3

Indicates whether the Aff3 field is valid. The defined values of this field are:

`0x80` Aff3 is valid.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field reads as `0x80`.

Aff1, bits [15:8], when level 0, or when level 1

Affinity level 1. The MPIDR_EL1.Aff1 field viewed from the highest Exception level of the associated PE or PEs.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Aff1, bits [15:8], when level 2, or when level 3

Indicates the Aff2 field is valid. The defined values of this field are:

`0x80` Aff2 is valid.

`0x00` Aff2 is not valid.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Aff0, bits [7:0], when level 0

Affinity level 0. The MPIDR_EL1.Aff0 field viewed from the highest Exception level of the associated PE or PEs.

Aff0, bits [7:0], when level 1, when level 2, or when level 3

Indicates the Aff1 field is not valid. The defined values of this field are:

`0x80` Aff1 is valid.

`0x00` Aff1 is not valid.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.16 ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the component. This allows software to make some use of a component that conforms to an architecture definition but whose part number is not necessarily recognized, or to disambiguate multiple architecturally-defined sub-components of a system.

Usage constraints

None.

Configurations

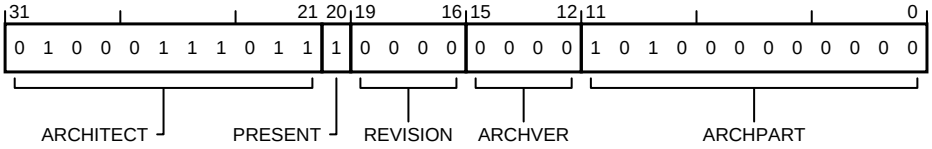
Always implemented.

Attributes

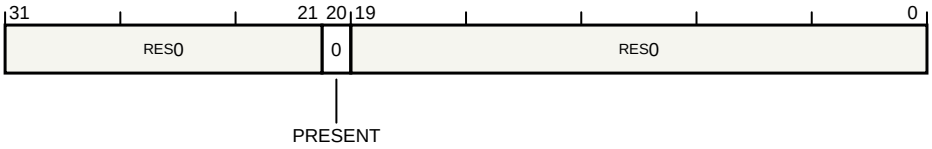
ERRDEVARCH is a 32-bit read-only memory-mapped register located at offset `0xFBC`.

Field descriptions

When DEVARCH is implemented, the ERRDEVARCH bit assignments are:



When DEVARCH is not implemented, the ERRDEVARCH bit assignments are:



Bits [31:21,19:0], when DEVARCH is not implemented

Reserved. This field is RES0.

ARCHITECT, bits [31:21], when DEVARCH is implemented

Architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code. The defined values of this field are:

`0x23B` JEP106 continuation code `0x4`, ID code `0x3B`. ARM Limited.

Other values are defined by the JEDEC JEP106 standard.

This field reads as `0x23B`.

PRESENT, bit [20]

DEVARCH present. Defines that the DEVARCH register is present. The defined values of this bit are:

`0` DEVARCH information not present. The rest of the register is RES0.

`1` DEVARCH information present.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

REVISION, bits [19:16], when DEVARCH is implemented

Defines the architecture revision. The defined values of this field are:

`0b0000` RAS system architecture v1.0.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field reads as `0b0000`.

ARCHVER, bits [15:12], when DEVARCH is implemented

Defines the architecture version of the component. The defined values of this field are:

`0b0000` RAS system architecture v1.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as `0b0000`.

ARCHPART, bits [11:0], when DEVARCH is implemented

Defines the architecture of the component. The defined values of this field are:

`0xA00` RAS system architecture.

Other values are reserved or define other component architectures.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as `0xA00`.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.17 ERRDEVID, Error Record Device ID Register

The ERRDEVID characteristics are:

Purpose

Defines the highest numbered index of the error records in this group.

Usage constraints

None.

Configurations

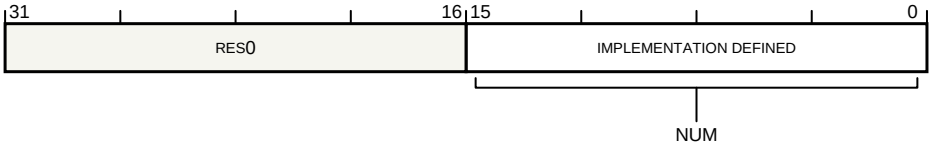
Always implemented.

Attributes

ERRDEVID is a 32-bit read-only memory-mapped register located at offset `0xFC8`.

Field descriptions

The ERRDEVID bit assignments are:



Bits [31:16]

Reserved. This field is RES0.

NUM, bits [15:0]

Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.18 ERRGSR<*n*>, Error Group Status Register

The ERRGSR<*n*> characteristics are:

Purpose

Each ERRGSR<*n*> register shows the status for up to 64 records in the group. <*n*> selects the set of 64 records from the records in the group.

Usage constraints

None.

Configurations

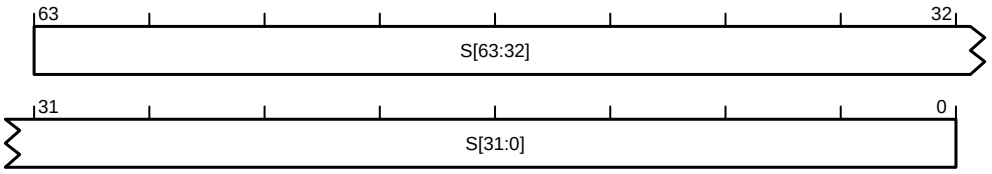
Always implemented.

Attributes

ERRGSR<*n*> is a 64-bit read-only memory-mapped register located at offset `0xE00` + 8×*n*.

Field descriptions

The ERRGSR<*n*> bit assignments are:



S[*m*], bit [*q*], for *q* = 0 to 63, where *m* = *q*+64×*n*

The status for Error Record <*m*>. A read-only copy of ERR<*m*>STATUS.V. The defined values of this bit are:

`0` No error.

`1` One or more errors.

This bit is RES0 if any of the following are true:

- <*m*> is greater than or equal to the number of implemented records.
- The record does not support this type of reporting.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.19 ERRIRQCR<*n*>, Generic Error Interrupt Configuration Register

The ERRIRQCR<0-15> characteristics are:

Purpose

The ERRIRQCR<*n*> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

Usage constraints

None.

Configurations

Present only if the interrupt configuration registers are implemented. RES0 otherwise.

Attributes

ERRIRQCR<*n*> is a 64-bit read/write memory-mapped register located at offset `0xE80` + 8×*n*.

Preface

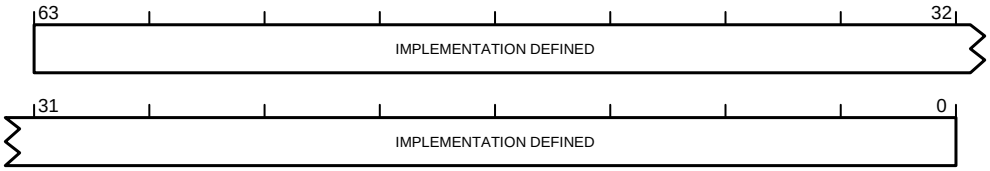
The architecture provides a recommended layout for the ERRIRQCR<*n*> registers. These registers are named ERRFHICR<0-2> and ERRERICR<0-2>, for the fault-handling and error-recovery interrupts respectively, and ERRIRQSR:

- ERRFHICR<*m*> map to ERRIRQCR0 and ERRIRQCR1.
- ERRERICR<*m*> map to ERRIRQCR2 and ERRIRQCR3.
- ERRIRQSR maps to ERRIRQCR15.

See ERR<irq>CR0, ERR<irq>CR1, ERR<irq>CR2, and ERRIRQSR. This section describes the generic, IMPLEMENTATION DEFINED, format.

Field descriptions

The ERRIRQCR<0-15> bit assignments are:



Bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.20 ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose

Recommended interrupt status register.

Usage constraints

None.

Configurations

Present only if the register locations for programming MSIs use the recommended layout. RES0 otherwise.

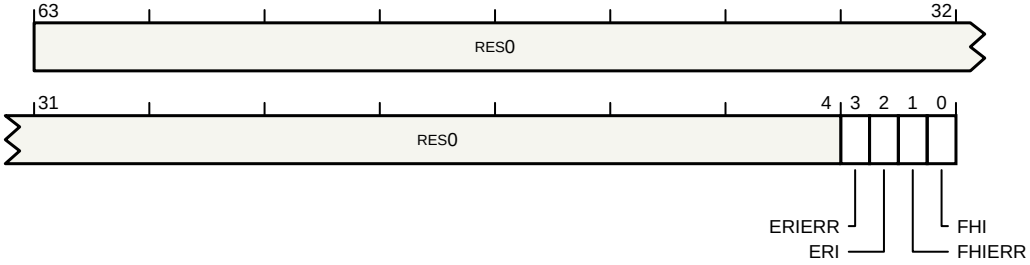
ERRIRQSR is architecturally mapped to ERRIRQCR31.

Attributes

ERRIRQSR is a 64-bit read/write memory-mapped register located at offset `0xEF8`.

Field descriptions

The ERRIRQSR bit assignments are:



Bits [63:4]

Reserved. This field is RES0.

ERIERR, bit [3]

Error Recovery Interrupt error. The possible values of this bit are:

`0` Error Recovery Interrupt write has not returned an error since this bit was last cleared to zero.

`1` Error Recovery Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

ERI, bit [2]

Error Recovery Interrupt write in progress. The defined values of this bit are:

`0` Error Recovery Interrupt write not in progress.

`1` Error Recovery Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

Note

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress. To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

FHIERR, bit [1]

Fault Handling Interrupt error. The possible values of this bit are:

`0` Fault Handling Interrupt write has not returned an error since this bit was last cleared to zero.

`1` Fault Handling Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

FHI, bit [0]

Fault Handling Interrupt write in progress. The defined values of this bit are:

`0` Fault Handling Interrupt write not in progress.

`1` Fault Handling Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

—— Note ——————————

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress. To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

————————————————

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.21 ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

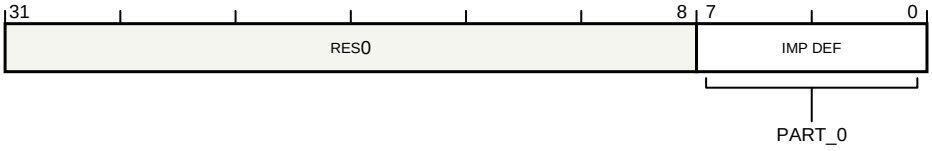
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

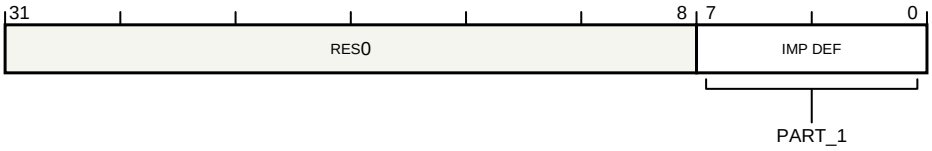
ERRPIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFE0`.

Field descriptions

When 12-bit part number, the ERRPIDR0 bit assignments are:



When 16-bit part number, the ERRPIDR0 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PART_0, bits [7:0], when 12-bit part number

Part number, bits [7:0]. This field reads as an IMPLEMENTATION DEFINED value.

PART_1, bits [7:0], when 16-bit part number

Part number, bits [11:4]. This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.22 ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

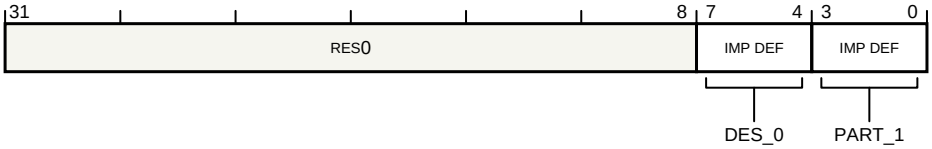
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

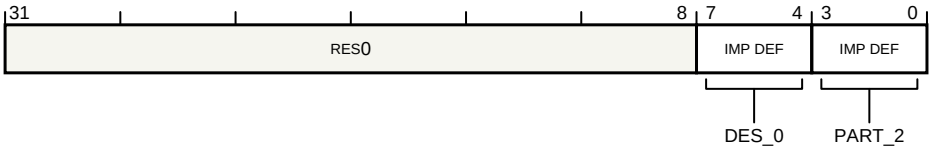
ERRPIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFE4`.

Field descriptions

When 12-bit part number, the ERRPIDR1 bit assignments are:



When 16-bit part number, the ERRPIDR1 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code bits [3:0]. This field reads as an IMPLEMENTATION DEFINED value.

PART_1, bits [3:0], when 12-bit part number

Part number, bits [11:8]. This field reads as an IMPLEMENTATION DEFINED value.

PART_2, bits [3:0], when 16-bit part number

Part number, bits [15:12]. This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.23 ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

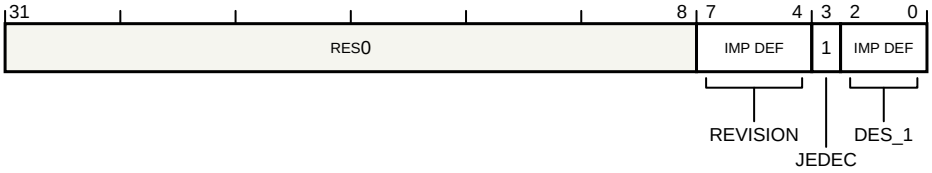
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

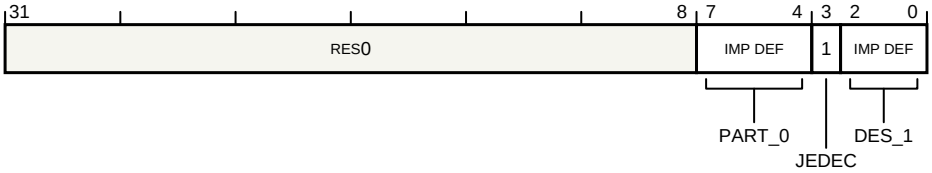
ERRPIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFE8`.

Field descriptions

When 12-bit part number, the ERRPIDR2 bit assignments are:



When 16-bit part number, the ERRPIDR2 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

REVISION, bits [7:4], when 12-bit part number

Component major revision. This field reads as an IMPLEMENTATION DEFINED value.

PART_0, bits [7:4], when 16-bit part number

Part number, bits [3:0]. This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]

Indicates that a JEDEC JEP106 identity code is used. This bit reads as one.

DES_1, bits [2:0]

Designer, JEP106 identification code bits [6:4]. The parity bit is not included.

This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.24 ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

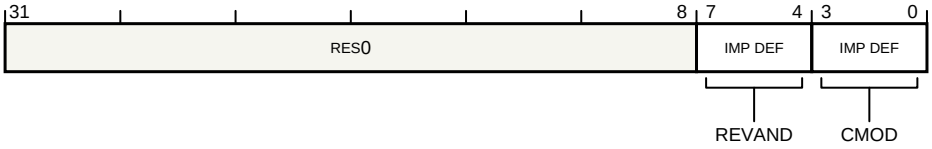
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

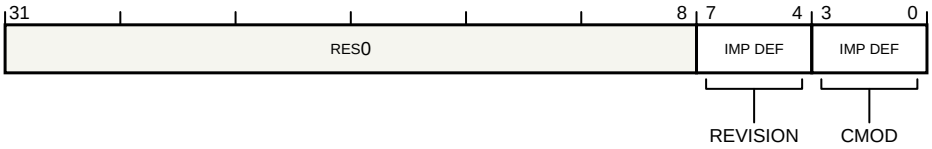
ERRPIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFEC`.

Field descriptions

When 12-bit part number, the ERRPIDR3 bit assignments are:



When 16-bit part number, the ERRPIDR3 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

REVISION, bits [7:4], when 16-bit part number

Component major revision. This field reads as an IMPLEMENTATION DEFINED value.

REVAND, bits [7:4], when 12-bit part number

Component minor revision. This field reads as an IMPLEMENTATION DEFINED value.

CMOD, bits [3:0]

Customer Modified. Indicates the component has been modified.

This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

4.6.25 ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

Usage constraints

None.

Configurations

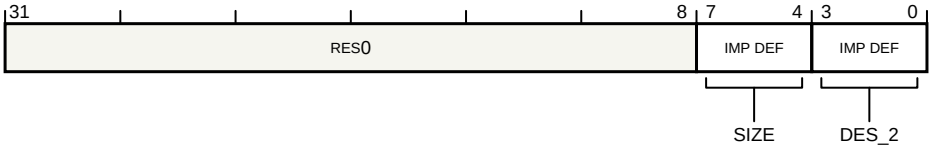
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRPIDR4 is a 32-bit read-only memory-mapped register located at offset `0xFD0`.

Field descriptions

The ERRPIDR4 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

SIZE, bits [7:4]

Size of the component. Log_2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

This field reads as an IMPLEMENTATION DEFINED value.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEP106 bank ID, minus 1.

This field reads as an IMPLEMENTATION DEFINED value.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

5 Appendix – Glossary

Asynchronous exception

In the ARMv8 architecture, an exception for which any of the following apply:

- The exception is not generated as a result of direct execution or attempted execution of the instruction stream.
- The return address presented to the exception handler is not guaranteed to indicate the instruction that caused the exception.
- The exception is imprecise.

Asynchronous exceptions are also known as interrupts.

Availability

Readiness for correct service.

Baseboard Management Controller

A PE dedicated to system control and monitoring.

BIST *Built-in self-test*

Built-in self-test

A mechanism that permits a machine to test itself.

Catastrophic failure

A failure with harmful consequences that are orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery.

CE *Correctable or Corrected Error*

Contained or containable error

An error that is not uncontained or uncontainable.

Containment

Limiting or preventing the silent propagation of an error. ARM recommends that the scope to which an error is contained is specified.

Correctable or Corrected Error

An error that is detected by hardware and that hardware can correct.

DECTED

Double error correct, triple error detect EDAC. This can detect a single, double or triple bit error and correct a single or double bit error in a protection granule.

Deferred error

An error that has not been silently propagated but does not require immediate action at the producer. The error might have passed from the producer to a consumer.

Detected error

An error that has been detected and signaled to a consumer.

Detected Uncorrectable Error

A detected error that cannot be corrected and causes failure.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Device memory

Memory locations where an access to the location can cause side-effects, or where the value returned for a load can vary depending on the number of loads performed. Typically, the Device memory attributes are used for memory-mapped peripherals and similar locations.

Double fault

A second error that is detected when the PE is in the process of handling a first error condition.

DUE *Detected Uncorrectable Error*

DUE FIT rate

The FIT rate for failures from a DUE.

ECC *Error Correction Code*

EDAC *Error Detection and Correction Code*

EDC *Error Detection Code*

Error

Deviation from correct service or a correct value.

Error Correction Code or Error Detection and Correction Code

A code capable of detecting and correcting a number of errors.

Error Detection Code

A code capable of detecting, but not correcting, errors.

Error propagation

Passing an error from a producer to a consumer.

Error record

Data recorded about an error, usually by hardware.

Error log

Historical data recorded about errors, usually by software.

Exception

An exception handles an event. For example, an exception could handle an external interrupt or an undefined instruction.

External abort

An in-band error that is generated as a response to a transaction. The name derives from the specific case of an abort generated by a memory system that is external to a PE, but the concept can apply to other interfaces.

In the context of the ARM architecture, this also refers to the type of exception generated when consuming an external abort.

Fail-safe

A failure mode in which the PE and other system components switch to backup mechanisms that keep processing instructions and data to allow either a safe shutdown or restart of the system, or to continue processing critical functions, or both.

Fail-secure

A failure mode in which the PE and other system components fail but the system is secured to allow either a safe shutdown or restart of the system, or to continue processing critical functions without exposing secret data, or both.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Fail-signaled
A failure mode in which the PE signals to the system that it has failed. It might continue to process instructions, but the system must ignore its output, or treat all outputs as detected errors.

Fail-silent
Failure mode in which the PE and all other system components (such as DMAs) stop processing instructions. A watchdog process will detect the failure and restart the system with an Error Recovery reset.

Failure
The event of deviation from correct service.

Failure In Time
The number of expected failures per billion hours of operation.

Fault
The cause of an error.

Fault injection
The deliberate injection of faults into a system for testing.

Fault prevention
Designing a system to avoid faults.

Fault removal
Logic or other mechanisms for detecting faults and correcting or bypassing their effect.

Field Replaceable Unit
The smallest unit that can be replaced without return to base.

FIT *Failure In Time*

FRU *Field Replaceable Unit*

General-purpose registers
The registers that the base instructions use for processing:

- In AArch32 state the general-purpose registers are R0-R14.
- In AArch64 state the general-purpose registers are R0-R30.

Generic Interrupt Controller
ARM system architecture interrupt controller for IRQ and FIQ interrupt exceptions.

GIC *Generic Interrupt Controller*

Hardware fault
A fault that originates in, or affects, hardware.

Imprecise exception
An exception that is not precise.

Infected
Being in error.

Interrupt
In a PE context, an asynchronous exception. There are three interrupt exceptions: IRQ, FIQ and SError. IRQ and FIQ are always precise.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

In a system architecture context, an asynchronous event sent to a PE or GIC for processing as an interrupt exception.

Isolation
Limiting the impact of an error only to components that actually try to use corrupted data.

Latent fault
An error that is present in a system but not yet detected.

MBIST
Memory BIST.

Minor failure
A failure with harmful consequences that are of a similar cost to the benefits that are provided by correct service delivery.

Normal memory
Used for bulk memory operations. Hardware might speculatively read these locations.

PE *Processing element*

Persistent fault
A fault that is not transient.

PFA *Predictive Failure Analysis*

Poisoned
State that has been marked as being in error so that subsequent consumption of the state will signal a detected error to a consumer.

Precise exception
An exception where the exception handler receives the state of the PE and the state of the memory system consistent with the PE having executed all of the instructions up to, but not including, the point in the instruction stream where the exception was taken. The state of the PE and the state of the memory do not include instructions that occurred after this point.

Predictive Failure Analysis
Mechanisms to analyze errors and predict future failures.

Processing element (PE)
The abstract machine defined in the ARMv8 architecture, as documented in an ARM Architecture Reference Manual. A PE implementation compliant with the ARMv8 architecture must conform with the behaviors described in the corresponding ARM Architecture Reference Manual.

Protection granule
A quantum of memory for which an EDC or ECC provides detection or correction. For example, a 72/64 SECDED ECC scheme has a 64-bit protection granule.

RAS *Reliability, Availability, Serviceability*

Recoverable error
A contained error that must be corrected to allow the correct operation of the system or smaller parts of the system to continue.

Reliability
Continuity of correct service.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Restartable error

A contained error that does not immediately impact correct operation. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part.

SDC *Silent Data Corruption*

SDC FIT rate

The FIT rate for failures because of SDC.

SDEC

Single device error correction EDAC. This can detect and correct multiple clustered errors in a protection granule, such as the types of errors that might be seen if a protection granule is striped across multiple devices and multiple errors come from a single device.

SECDED

Single error correct, double error detect EDAC. This can detect a single or double bit error and correct a single bit error in a protection granule.

SED

Single error detect EDC. This can detect a single bit error in a protection granule.

SError Interrupt

An asynchronous interrupt in the ARMv8 architecture.

Serviceability

The ability to undergo modifications and repairs.

Service failure mode

A mode entered to reduce the severity of an error.

Silent Data Corruption

An error that is not detected by hardware or software.

Silently propagated

An error that is passed from place to place without being signaled as a detected error.

Software fault

A fault that originates in and affects software.

System Control Processor

A PE dedicated to system control and monitoring.

Synchronous exception

In the ARMv8 architecture, an exception for which all of the following apply:

- The exception is generated as a result of direct execution or attempted execution of an instruction.
- The return address presented to the exception handler is guaranteed to indicate the instruction that caused the exception.
- The exception is precise.

Synchronous External Abort

A synchronous exception in the ARMv8 architecture.

Transient fault

A fault that is not persistent.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential

Uncontained or uncontainable error

An error that has been, or might have been, silently propagated.

Undetected fault

See *Latent fault*.

Unrecoverable error

A contained error that is not recoverable. Continued correct operation is generally not possible. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part. Systems might use high-level recovery techniques to work around an unrecoverable yet contained error in a component so that the system recovers from the error.

Copyright © 2017 ARM Limited or its affiliates. All rights reserved.
Non-Confidential