Arm® Reliability, Availability, and Serviceability (RAS) System Architecture

for A-profile architecture

Document number	IHI0100
Document quality	EAC
Document version	A.b
Document confidentiality	Non-Confidential

Document number: IHI0100

Release information

The following releases of this document have been made.

Date	Version	Changes
30/Nov/2025	A.b	• Maintenance release of the RAS System Architecture document with updates to rules and the RAS Memory-mapped registers.
16/Jan/2025	A.a	• Initial release of the RAS System Architecture standalone document.

Arm Non-Confidential Document License ("License")

This License is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

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

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

THE 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. Arm may make changes to the Document at any time and without notice. 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, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary

hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

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.

The information in this supplement is at EAC quality, which means that all features of the specification are described in the supplement.

Web Address

https://www.arm.com

Contents

Arm® Reliability, Availability, and Serviceability (RAS) System Architecture, for A-profile architecture

Preface

This preface introduces the *RAS System Architecture, for A-profile architecture*. It contains the following sections:

- About this Document.
- Using this Document.
- Conventions.
- Additional reading.
- Feedback.

About this Document

This Document describes the Arm® RAS System Architecture.

This Document is organized into chapters:

Chapter 1, Introduction to RAS

Provides an introduction to the RAS System Architecture. Defines terminology used regarding the RAS System Architecture.

Chapter 2, RAS System Architecture

Describes the RAS System Architecture. It includes details of nodes, the standard Error record, Error recovery interrupts, Fault handling interrupts, and Critical error interrupts. It also describes extensions of the RAS System Architecture, some of which are optional.

Chapter 3, RAS Memory-mapped Register Descriptions

Describes the RAS System Architecture registers.

Using this Document

This Document is intended to be read in conjunction with the *Arm® Architecture Reference Manual, for A-profile architecture*.

Conventions

The following sections describe conventions that this book can use:

- Typographic conventions.
- Rules-based writing.
- Signals.
- Numbers.
- Pseudocode descriptions.

Typographic conventions

The typographical conventions are:

italic Introduces special terminology, and denotes citations.

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

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

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

SMALL CAPITALS

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

Colored text Indicates a link. This can be:

- A URL, for example https://developer.arm.com.
- A link, to a chapter or appendix, or to a glossary entry, or to the section of the Document that defines the colored term, for example Software fault or ERRIIDR.

{ and } Braces, { and }, have two distinct uses:

Optional items

In syntax descriptions braces enclose optional items. In the following example they indicate that the `<shift>`parameter is optional:

```
ADD <Wd|WSP>, <Wn|WSP>, #<imm>{, <shift>}
```

Similarly, they can be used in generalized field descriptions, for example ERRACR.{N}SRA refers to a field in the ERRACR register that is called either NSRA or SRA.

Sets of items

Braces can be used to enclose sets. For example, ERR<n>STATUS.{RV, RV2} refers to a set of two register fields, ERR<n>STATUS.RV and ERR<n>STATUS.RV2.

Notes Notes are formatted as:

Note
This is a Note.

In this Document, Notes are used only to provide additional information, usually to help understanding of the text. While a Note might repeat architectural information given elsewhere in the Document, a Note never provides any part of the definition of the architecture.

Rules-based writing

Some sections of this Document use rules-based writing. Rules-based writing consists of a set of individual content items. A content item is classified as one of the following:

- Rule.
- Information.
- Software usage.
- Declaration.

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

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

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

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

Content item identifiers

A content item may have an associated identifier which is unique among content items in this Document. After content reaches beta status, a given content item has the same identifier across subsequent versions of this Document.

Content item rendering

In this Document, a content item is rendered with a token of the following format in the left margin: L_{iiii}. L is a label that indicates the content class of the content item. iiii is the identifier of the content item.

Content item classes

Each of the content item classes has a different function in this Document.

Rule

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

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label R.

Information

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

An Information statement is rendered with the label I.

Software usage

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

A Software usage statement is rendered with the label S.

Declaration

A Declaration statement introduces concepts or terminology.

A Declaration does not describe behavior.

A Declaration is rendered with the label D.

Signals

In general this specification does not define hardware signals, but it does include some signal examples and recommendations. The signal conventions are:

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

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

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

Numbers

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

Pseudocode descriptions

This Document uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in `monospace` font.

Additional reading

This section lists relevant publications from Arm.

See Arm Developer, https://developer.arm.com, for access to Arm documentation.

Arm publications

- *Arm® Architecture Reference Manual for A-profile architecture* (ARM DDI 0487).

Feedback

Arm welcomes feedback on its documentation.

Feedback on this Document

If you have any comments or queries about this Document, create a ticket at https://support.developer.arm.com.

As part of the ticket, include:

- The title, Arm® Reliability, Availability, and Serviceability (RAS) System Architecture, for A-profile architecture.
- The number, IHI0100 A.b.
- The section name to which your comments refer.
- The page number(s) to which your comments refer.
- The rule identifier(s) to which your comments refer, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader. Search performance may be significantly improved by increasing the Fast Find Maximum Cache Size under Search in Preferences.

Inclusive terminology commitment

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

We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact `terms@arm.com`.

Chapter 1
Introduction to RAS

This chapter introduces the RAS System Architecture. It contains the following sections:

- Introduction.
- Faults, errors, and failures.
- General taxonomy of errors.
- Techniques for improving reliability, availability, and serviceability.

1.1 Introduction

I_{LMHPD} RAS are three aspects of the *dependability* of a system:

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

I_{HWHJM} 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.

I_{PQNZV} This Document describes the RAS System Architecture. For information on RAS PE architecture, see the chapter *RAS PE architecture* in *Arm® Architecture Reference Manual, for A-profile architecture*.

1.2 Faults, errors, and failures

R_{NVNNC} Correct service is delivered when the service implements the system function.

I_{QWSVK} Correct service might include:

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

R_{KRTSC} For the purpose of describing the RAS System Architecture, 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 from correct service. An incorrect value that has an error is corrupt.
- A fault is the cause of the error.

R_{JNBDX} Errors that are present but not detected are latent errors or undetected errors.

I_{TNQPK} In a system with no error detection, all errors are latent errors and are silently propagated by components until they are either masked or cause failure.

I_{GRYKV} 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.

I_{NMGPQ} 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 System Architecture primarily addresses errors produced from hardware faults. These fall into two main areas:

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

1.3 General taxonomy of errors

1.3.1 Error detection

R_{FHXWP} When a component accesses memory or other state, an error might be detected in that memory or state.

I_{WKPVR} The error might be corrected or deferred by the component, or signaled to another component as either a deferred error or a detected error.

1.3.2 Error propagation

R_{LRZDN} A transaction occurs when a producer of the transaction passes a value or other signal to a consumer of the transaction.

I_{VYCCX} Transactions are part of the service provided by the producer for the consumer.

I_{RHGDL} In many protocols and service interface definitions, a high-level transaction consists of a sequence of operations, for instance between a *Requester* and a *Completer*.
For the purposes of this manual, the most basic form of a unidirectional transfer between a *producer* and *consumer* is considered as a transaction.
That is, each one of the sequence of operations is considered a separate transaction. For some operations, such as a request, the Requester is producer and the Completer is the consumer. For other operations, such as a response, the Completer is producer and the Requester is the consumer.

R_{SKZZG} An error is propagated by the producer of a transaction when the service interface is incorrect because of the error. The error is propagated to the consumer.

R_{CHCCV} An error is propagated by deviations from correct service, including when any of the following occurs that would not have been permitted to occur had the fault not been activated:

- A corrupt value is passed from producer to consumer.
- A transaction or other operation occurs that should not have occurred.
- A transaction or other operation that should have occurred does not occur.
- A loss of uniprocessor semantics or any other loss of coherency in a multiprocessor coherent system is observed.
- Changing the timing and/or order of transactions or other operations such that the timing and/or order of those transactions or operations is incorrect. In this case, the service interface defines acceptable timings and/or orders for transactions and other operations.

R_{YFBRV} The service interface for a transaction might include means to signal that the transaction is propagating either of the following:

- A detected error.
- A deferred error.

R_{BHWVX} An error is silently propagated by the producer of a transaction if the consumer of the transaction cannot detect the error and consumes an undetected error because of the transaction. This might be because of one of the following:

- The error is present on the transaction, but was not detected by the producer. The error is silently propagated by the producer.
- The error is present on the transaction, but was not signaled to the consumer as an error. For example, a corrupt value was passed in the transaction with no indication that it was corrupt. The error is silently propagated by the producer.

R_{FPBYS} A latent, possibly detectable, error is silently propagated by the consumer of an otherwise correct transaction if the transaction causes the error to become undetectable.

Example 1-1

A partial write to a protection granule removes poison, leaving the unchanged portion of the location corrupt. To implement a partial write, the consumer logically reads the current value of the location, modifies the value, and then writes the modified value back. These are internal transactions in the consumer that silently propagate the error. In this example there was no error at the producer nor on the transaction.

I_{QXRLB} Errors might be propagated by components in a system until one of the following occurs:

- They are masked and do not affect the outcome of the system.
  The error might be masked because a corrupt value is discarded or overwritten, or the error is detected and removed.
- They affect the service interface of the system and possibly cause failure. If the error has been silently propagated to the service interface, then:
  - This is a Silent Data Corruption, SDC.
  - The rate of such failures, measured as the number of failures per billion device-hours of operation, is called the Silent Data Corruption Failure-in-Time rate, SDC FIT rate.

  Alternatively, the error might have been detected, causing the system to invoke error handling and recovery. See Error handling and recovery.

1.3.3 Infected and poisoned

R_{KNHWB} The state of a component becomes infected when the component consumes an uncorrected error that updates the state.

R_{TZBSW} A value is poisoned in the state of a component if it is marked as being in error, such that a subsequent access of the state will detect the value is so marked and is treated as a detected error.

I_{YBMFK} Poison is used to defer an error.

1.3.4 Containable and uncontainable

R_{DXQRD} An undetected error is uncontained at the component that failed to detect it.

R_{RJYRQ} A silently propagated error is uncontained at the component that silently propagated it.

R_{GJQNR} A Detected Uncorrected Error is uncontainable at the component if it might be uncontained at the component. A Detected Uncorrected Error is containable at the component if it is not uncontainable at the component. If the component cannot determine whether a Detected Uncorrected Error is uncontainable at the component or containable at the component, then the component treats the Detected Uncorrected Error as uncontainable at the component.

I_{MRDMR} An error that is uncontainable at the component might be containable at the system level.

I_{NWZGB} Reporting an error as containable allows software to contain the error. This does not mean that hardware has contained the error.

1.4 Techniques for improving reliability, availability, and serviceability

I_{TPGKF} Each device sets its own targets for reliability, availability, and serviceability using various techniques to achieve these targets, including:

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

I_{DMKGY} The level of reliability, availability, and serviceability in any implementation, and which parts of the system include RAS, are IMPLEMENTATION DEFINED. The RAS Extension and RAS System Architecture do not prescribe the level of reliability, availability, and serviceability in any implementation, or which parts of the system include RAS.

1.4.1 Fault prevention and fault removal

R_{YLVTS} Fault prevention and fault removal are two techniques for handling faults. Fault prevention and fault removal mechanisms are IMPLEMENTATION DEFINED.

I_{WZTKF} Fault prevention techniques are outside the scope of the architecture.

I_{JVLNC} Some errors can be corrected. A corrected error might be recorded and notified to software, for example by a Fault handling interrupt, but it is not propagated. This means that it is not consumed and does not cause service failure.
A corrected error might be used to remove the original fault, for example by writing the corrected value back to the fault location, or the fault might remain in place after corrected value is returned to the requester.

I_{WSPBC} A common technique to detect and correct errors is the use of an *Error Detection and Correction Code* (EDAC), more commonly referred to as simply an *Error Correction Code* (ECC). ECC schemes use mathematical codes to detect and correct an error in a value in memory. The size of the value is the protection granule for the ECC scheme.

I_{PBJLC} The RAS Extension and RAS System Architecture do not require implementation of any fault removal schemes, including ECC.

1.4.2 Error handling and recovery

R_{XPLVT} An error that is not corrected is an uncorrected error.

R_{VTXYY} Error recovery is the process by which software and hardware minimize the impact of an uncorrected error.

I_{LYWFS} Error recovery methods include all of the following:

- Deferring an error from a fault. An error is deferred by hardware if hardware can make forward progress without consuming the error.
  Deferring the error means:
  - The fault might become masked later (fault removal). For example, because the corrupt value is overwritten before it is consumed.
  - If the deferred error is later consumed, then the error is reported at the point of consumption. For example, if the deferred error is consumed by a PE then the consumer PE generates an Error exception. This can give better results in terms of error recovery in the case where the original producer of the data is not known when the error was deferred. For example because a latent error was detected.

  A common technique to defer an error is to replace the corrupt value with a poisoned value, for example in memory or in a transaction.
- Preventing further propagation of the error, that is containing the error. In particular, preventing silent propagation of the error.
- Reducing the severity of a failure by invoking a service failure mode:
  - This is a *Detected Uncorrected Error* (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.

I_{BRDMK} A software error recovery agent is typically invoked when hardware detects an error it cannot correct, defer, or remove.

I_{PGXFK} An error recovery agent also provides information to the operator through error logs to improve serviceability, for example to help with the identification of a Field Replaceable Unit, FRU.

I_{MFPRY} The RAS Extension and RAS System Architecture provide optional common programmers' models to record information about an error in an Error record.

I_{CVFFN} The RAS Extension describes the behavior of a PE when an error is signaled to it by the system, including invoking a service failure mode by taking an Error exception, and optional mechanisms to limit propagation of an error.

I_{TLDCY} The RAS Extension and RAS System Architecture do not require systems to implement error recovery mechanisms, including poison, and do not require systems to limit the silent propagation of errors.

1.4.3 Fault handling

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

I_{GGCDN} Fault handling methods include *Predictive Failure Analysis* (PFA), using information recorded by hardware to trigger preemptive action.

I_{WNHJF} The RAS Extension and RAS System Architecture provide optional mechanisms to allow the reporting of errors and warnings to a fault handling agent, and to record information about the fault in 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*.

I_{FQRSQ} The detailed nature of the fault handling agent is outside the scope of this architecture. Fault handling and Error recovery might be independent agents.

I_{DQBCJ} See also Standard error record.

Chapter 2
RAS System Architecture

This chapter describes the RAS System Architecture. It contains the following sections:

- About the RAS System Architecture.
- Nodes.
- Detecting and consuming errors.
- Standard error record.
- RAS interrupts.
- In-band error response signaling
- Error record reset.
- Extensions.
- Accessing RAS registers.

2.1 About the RAS System Architecture

I_{XKHGG} The RAS System Architecture provides a framework for building RAS features in a system. It provides a reusable component architecture for components that can detect and record errors, and signal them to a PE.

R_{DKJPB} A node is a RAS System Architecture element that records errors detected or consumed by one or more system components.

I_{NTRXQ} A RAS System Architecture implementation includes one or more nodes. The RAS System Architecture does not require that all components in a system implement the RAS System Architecture or appear as a node.

I_{FPMKF} The RAS System Architecture does not prescribe the level of reliability, availability, and serviceability in the system. The RAS features that the system includes, for example to detect, correct, contain, or defer errors, are IMPLEMENTATION DEFINED.

I_{LJWMZ} The RAS features and behavior of components that do not implement the RAS System Architecture are IMPLEMENTATION DEFINED.

I_{QTZCK} Arm recommends that all errors are reported to a RAS System Architecture node to enable error recovery and fault handling.

I_{HTDRT} This section describes the behavior of RAS System Architecture nodes, and other required behaviors of components that implement the RAS System Architecture.

D_{TGKPM} In this chapter, OPTIONAL *from* or *permitted from* refers to the earliest version of the RAS System Architecture in which the feature is permitted to be implemented. A system must be FEAT_RASSAv1p1 compliant to include a feature permitted from FEAT_RASSAv1p1, and must be FEAT_RASSAv2 compliant to include a feature permitted from FEAT_RASSAv2. If no version is indicated, this means the feature is OPTIONAL from or permitted from FEAT_RASSAv1.

R_{BBJHD} Unless otherwise specified, all required features in FEAT_RASSAv1 compliant implementations are required in FEAT_RASSAv1p1 compliant implementations, and all required features in FEAT_RASSAv1p1 compliant implementations are required in FEAT_RASSAv2 compliant implementations.

R_{ZPMBK} Except where restricted by D_{TGKPM}, FEAT_RASSAv1 compliant implementations can implement any subset of features from FEAT_RASSAv1p1, and FEAT_RASSAv1p1 compliant implementations can implement any subset of features from FEAT_RASSAv2.

2.2 Nodes

I_{RDHHP} A component might implement one or more nodes, or a node might be implemented outside of a component.
See also R_{WXPDN} and R_{GCDCL}.

D_{BYMCW} The RAS System Architecture defines the following features for nodes:

Error detection and correction

The level of error correction and detection implemented at a component is IMPLEMENTATION DEFINED.

A node might include the control to disable error reporting and recording of detected errors, for example while software initializes the component.

It is IMPLEMENTATION DEFINED whether error detection and correction is fully disabled at the component when reporting and recording are disabled at the node.

See Detecting and consuming errors.

Records

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

See Standard error record.

Fault handling interrupt

Asynchronous (out-of-band) reporting of all or some recorded errors by an interrupt. The fault handling interrupt can report corrected errors, deferred errors, and uncorrected errors. It is IMPLEMENTATION DEFINED whether a node provides a single control for all errors, or separate controls for each type of error.

See Fault handling interrupt.

Error recovery interrupt

Asynchronous (out-of-band) reporting of recorded 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 the control to enable deferred errors to be reported in this way. If the control is not provided, then deferred errors are not reported by this means.

See Error recovery interrupt.

Critical Error interrupt

Critical error interrupts provide a mechanism for a node to report a critical error condition to a system controller for error recovery.

See Critical error interrupt.

In-band error response (External abort)

In-band signaling of a detected uncorrected error to the Requester of the transaction. This is also referred to as an External abort.

Corrected errors and errors deferred to the Requester are not reported by such means.

See In-band error response signaling.

Timestamps

It is IMPLEMENTATION DEFINED whether a node records a timestamp in each Error record.

See The RAS Timestamp Extension.

Corrected error counter

It is IMPLEMENTATION DEFINED whether a node implements a counter for counting errors. Software can poll the error counter or initialize the counter with a threshold value and receive an interrupt when the counter overflows. A counter overflows when incrementing the counter results in unsigned integer overflow.

It is IMPLEMENTATION DEFINED which corrected errors are counted.

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

See Standard format Corrected error counter.

Proxies

A node can be a proxy for another component implementing multiple other nodes. In this case, there is a proxy error record.

See System RAS Agents.

I_{WRWMK} A node might implement some or all of these features.

R_{YHBGJ} The first standard Error record for a node contains:

- An identification register, ERR<n>FR, that describes the implemented features of the node.
- The ERR<n>CTLR register to enable or disable the features.

R_{JMRML} A node has a single ERR<n>FR and a single ERR<n>CTLR register.

R_{CWWXN} If the node implements multiple Error records, then each Error record has the same features and all Error records share the controls.

Note

If a component requires multiple sets of controls, then the component implements multiple nodes.

R_{GSGNZ} 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.

2.2.1 Multiple error records per node

R_{RMRKT} Each node contains at least one Error record.

I_{YKNTD} A node might implement multiple Error records for one or more of the following purposes:

- To record different types of error in different Error records.
- To record errors from different components, or different FRUs accessed by a component, in different Error records.
- To record multiple errors.

R_{PZCQV} If a single node implements multiple Error records, then all of the following are true:

- The Error records are indexed sequentially within an Error record group starting from the first Error record for the node.
- For each Error record other than the first Error record for the node, the following are true:
  - The ERR<n>FR.ED field is `0b00`.
  - If FEAT_RASSA_ERT is not implemented, ERR<n>FR[63:2] are RES0.
  - The ERR<n>CTLR register is RES0.

D_{HFVSH} When FEAT_RASSAv2 is implemented, each node implements support for FEAT_RASSA_ERT. FEAT_RASSA_ERT is permitted from FEAT_RASSAv2.

I_{CCZLX} FEAT_RASSA_ERT enables software to discover differences between error records owned by the same error node, and allows *continuation records*, which enable an error node to record additional IMPLEMENTATION DEFINED syndrome information.

I_{KVCGG} In a continuation record, error record <n> is a continuation of error record <n-1>. Error record <n−1> might also be a continuation of error record <n−2>, and so on.

D_{DJPDM} When FEAT_RASSA_ERT is implemented, if ERR<n>FR.ED is `0b00`, then ERR<n>FR.ERT defines the *error record type*.

R_{BRDWX} Continuation records are only permitted in an Error record group. Within the Error record group, error record 0 is not permitted to be a continuation record.

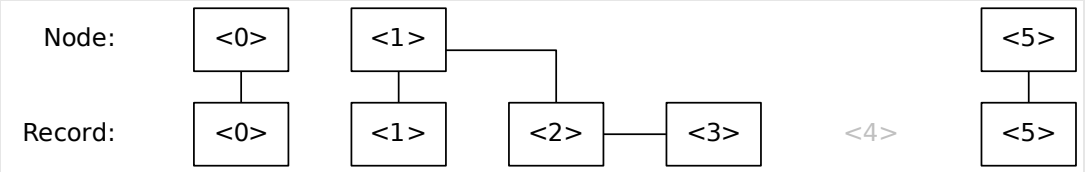
R_{RFPVW} An Error record group consists of the Error records of one or more nodes.

R_{DBPFH} An Error record group might be sparsely populated. Locations relating to unimplemented Error records are RAZ/WI, meaning that they have an ERR<n>FR register that reads as zero.

See Nodes.

Example 2-1

An Error record group contains five error records owned by three nodes, arranged as shown below:



- Node <0> owns a single Error record: <0>. ERR0FR describes the features for this node, and ERR0CTLR contains the controls for this node. ERR0STATUS, ERR0ADDR, and ERR0MISC<m> record syndrome for this Error record.
- Node <1> owns three Error records: <1>, <2>, and <3>.
  - Error record <1> is the first error record of the node. ERR1FR.ED is `0b01` or `0b10`. ERR1FR describes the features for this node, and ERR1CTLR contains the controls for this node. ERR1STATUS, ERR1ADDR, and ERR1MISC<m> record syndrome for this Error record.
  - ERR2FR.{ED, ERT} is {`0b00`, `0b00`} and ERR2CTLR is RES0. ERR2STATUS, ERR2ADDR, and ERR2MISC<m> record syndrome for this Error record.
  - Error record <3> is a continuation of Error record <2>. ERR3FR.{ED, ERT} is {`0b00`, `0b01`}, ERR3CTLR is RES0, ERR3STATUS.{V, AV, MV, IERR} are defined and all other values in ERR3STATUS are RES0, and ERR3ADDR and ERR3MISC<m> are IMPLEMENTATION DEFINED, recording additional syndrome for the error recorded by Error record <2>.
- Error record <4> is not implemented. ERR4FR.{ED, ERT} is {`0b00`, `0b00`}, and ERR4CTLR, ERR4STATUS, ERR4ADDR, and ERR4MISC<m> are RAZ/WI.
- Node <5> owns a single error record: <5>. ERR5FR describes the features for this node, and ERR5CTLR contains the controls for this node. ERR5STATUS, ERR5ADDR, and ERR5MISC<m> record syndrome for this Error record.
- If the Error record group is accessed using a memory-mapped view then ERRDEVID.NUM is 6.
- If the Error record group is accessed using System registers then ERRIDR_EL1.NUM is 6.

2.3 Detecting and consuming errors

R_{QZHDT} A component detects an error when it detects that a deviation from correct service has occurred or will occur. For example, including but not limited to when any of the following occurs that would not be permitted to occur had the fault not been activated:

- A corrupt value has been or will be passed to a consumer.
- A transaction or other operation occurs or will occur that should not occur.
- A transaction or other operation that should occur does not occur or will not occur.
- A loss of uniprocessor semantics or any other loss of coherency in a multiprocessor coherent system is or will be observed. See I_{SVZKY}.
- The timing and/or order of transactions or other operations has been or will be changed.
- A latent error has become or will become undetectable. See I_{QXPLK}.

I_{SVZKY} Examples of a loss of uniprocessor semantics or other loss of coherency that might occur because of an error include:

- A cache loses data that it holds in a modified state.
- A cache writes back unmodified data to memory.

An example that should not occur is when a partial write to the protection granule of a cache location holding poison occurs, and the cache later invalidates the line without writing back the poison value.

Example 2-2 Maintaining poisoned cache locations

A cache fetches data from memory and receives poison, and subsequently, a partial write to that location is insufficient to clean the location of the poison and the location remains poisoned.

The cache should treat the location as modified, even though it appears that the write did not modify the location.

That is, the cache should take ownership of the location and write-back poison when the location is evicted from the cache. Otherwise if the original error was transient and later disappears from memory, the location reverts to the unmodified value, silently propagating the error.

I_{QXPLK} An example of a latent error becoming undetectable includes when a poison value indicating a deferred error is lost at the interface between domains. For example, because a poison value is passed to a component that does not support poisoning.

An example of a latent error becoming undetectable that should not occur is when a poison value is lost by a partial write to the protection granule. In this case, the partial write should leave the protection granule containing poison.

R_{LRSMZ} A component consumes an error that is signaled to the component in response to a memory access, cache maintenance operation, or other transaction initiated by the component as one of:

- An In-band error response.
- A deferred error.

R_{WXPDN} When an error is detected or consumed by a component, the error is reported to one or more nodes.

R_{VYRXT} It is IMPLEMENTATION DEFINED whether a Requester that consumes a signaled detected error reports the consumed error.

R_{LRQSG} It is IMPLEMENTATION DEFINED whether errors are reported when a detected error is propagated between components.

R_{WDJGD} It is IMPLEMENTATION DEFINED whether all corrected errors are reported.

R_{GVPMK} It is IMPLEMENTATION DEFINED whether errors detected on hardware speculation are reported.

R_{GCDCL} It is IMPLEMENTATION DEFINED whether the node or nodes that an error is reported to are one or more of the following:

- The same component that detected the error.
- The consumer of the transaction that consumes a detected error signaled by the producer of the transaction which detected the error. Syndrome information might be passed with the signaled detected error to the consumer.

- Another component that neither detected nor consumed the error. For example, a node whose purpose is to record errors for other components. Such a node might comprise one record for each component for which it is recording an error, or a number of shared records, where each record identifies the originating component, or some other arrangement.

R_{LBHMF} When an error is detected or consumed by a component, if the error can be corrected:

- The error is corrected.
- Optionally, the detected error is reported to a node, the node records the component error state as Corrected, and if implemented and enabled, a Fault handling interrupt is raised.
- If the error is detected on a read access by a Requester, corrected data is returned to the Requester.

R_{LMCVC} When an error is detected or consumed by a component, if the error cannot be corrected and can be deferred:

- The error is deferred. For example, the location being accessed is poisoned or poisoned data is returned to the Requester.
- The error is reported to a node and the node records the component error state as Deferred.
- If the error is detected on an access by a Requester, the error is not deferred to the Requester, and if implemented and enabled, it is IMPLEMENTATION DEFINED whether an In-band error response is returned to the Requester.
- If the error is detected on a read access by a Requester, the error is not deferred to the Requester, and an In-band error response is not returned to the Requester, the data returned to the Requester is IMPLEMENTATION DEFINED and might be UNKNOWN.
- If implemented and enabled, a Fault handling interrupt is raised.
- If implemented and enabled, an Error recovery interrupt is raised.

Note

An error cannot be deferred to a component that does not accept deferred errors.

R_{LKCNC} When an error is detected or consumed by a component, if the error cannot be corrected and cannot be deferred:

- The error is reported to a node and the node records the component error state as Uncorrected.
- If implemented and enabled, a Fault handling interrupt is raised.
- If implemented and enabled, an Error recovery interrupt is raised.
- If the error is detected on an access by a Requester, and if implemented and enabled, an In-band error response is returned to the Requester.
- If the error is detected on a read access by a Requester, and an In-band error response is not returned to the Requester, the data returned to the Requester is IMPLEMENTATION DEFINED and might be UNKNOWN.
- If the component is unable to continue operation, it might enter a service failure mode.

I_{NJHPF} The criteria by which a component determines when it can correct or defer an error are IMPLEMENTATION DEFINED. For example, if the error is detected in response to an access by a Requester that is not capable of receiving a deferred error response, then it is not possible to defer the error to the Requester.

I_{QQRKD} R_{LMCVC} permits a component to both defer an error and return an In-band error response to the Requester. For instance if it is not possible to defer the error to the Requester.

Example 2-3 Treating poisoned PE caches

A PE executes a load instruction which misses in the PE cache and the subsequent cache refill receives poison in the cache line for the location being accessed. The cache line is allocated into the cache, but the cache cannot return poison to PE and signals an In-band error response to the PE. It is IMPLEMENTATION DEFINED whether the cache records this with a component error state of Deferred or Uncorrected.

I_{LRNRJ} R_{LKCNC} and R_{LMCVC} permit a component to return a fixed known value to a Requester when an uncorrected error is detected on a read access, not deferred to the Requester, and either support for an In-band error response is not implemented or the In-band error response is disabled. For example, zero or an all-ones value.

See also Software faults.

R_{LTBDP} When an error is reported to a node, the node records syndrome information for the error in a standard Error record.

I_{SNNZR} Arm recommends that hardware records sufficient information to:

- Determine whether error recovery is possible, if the error was not corrected by hardware.
- Allow fault analysis to find trends in the faults. This information is IMPLEMENTATION DEFINED but might include the location of the data.
- Allow identification of a FRU.

I_{JNMFY} The node registers might also contain control registers for error detection, correction and reporting at the component.

I_{WMVTN} Corrected errors can be recorded by counting each corrected error. Counting might be done by either software or hardware. The fault handling process compares the corrected error count or rate with a threshold value to determine whether to take action.

I_{QGNHF} Standard format Corrected error counter and Corrected error counter describe an optional standard hardware mechanism for counting errors.

I_{GGQSR} The details of any service failure mode are IMPLEMENTATION DEFINED. For example:

- A component that fetches data from memory and processes that data might halt processing and await servicing by an application processor when it receives an In-band error response. This is a form of service failure mode.
- When a PE takes an Error exception and executes an error handler, this is also a form of service failure mode.

The component might implement multiple functions, some of which can be in a service failure mode while others continue to operate, or the service failure mode might affect multiple or all functions of the component.

I_{ZZKRS} See also:

- Standard error record.
- Fault handling interrupt.
- Error recovery interrupt.
- In-band error response signaling.
- Standard format Corrected error counter.

2.4 Standard error record

R_{GTCQJ} The RAS System Architecture defines a standard Error record and a mechanism to access Error records as System registers or as a memory-mapped component.

I_{BNPZB} The format of the Error record registers is the same for both access mechanisms.

R_{XGGTZ} The standard Error record contains:

- 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.
- IMPLEMENTATION DEFINED status registers, referred to as ERR<n>MISC<m>. Arm recommends these are used for:
  - Identifying a FRU.
  - Locating the error within the FRU.
  - Optionally, a Corrected error counter or counters for software to poll the rate of corrected errors.
  - Optionally, a timestamp value for when the error was recorded.

R_{MQPFL} When FEAT_RASSAv1 is implemented, there are two ERR<n>MISC<m> registers for each Error record:

- ERR<n>MISC0.
- ERR<n>MISC1.

R_{QCKVG} When FEAT_RASSAv1p1 is implemented, there are four ERR<n>MISC<m> registers for each Error record:

- ERR<n>MISC0.
- ERR<n>MISC1.
- ERR<n>MISC2.
- ERR<n>MISC3.

I_{PSZMK} The RAS System Architecture permits the implementation of ERR<n>MISC2 and ERR<n>MISC3 in implementations of the FEAT_RASSAv1.

R_{DXZPX} An Error record might include additional IMPLEMENTATION DEFINED controls and identification registers.

I_{PBJTL} Accessing RAS registers defines reusable formats for a memory-mapped views of Error records. Use of reusable formats by any component in the system is OPTIONAL.

R_{WDSFZ} Error records are preserved over an Error Recovery reset. This allows for a diagnosis after system failure.

2.4.1 Additional Error record types

R_{KWYKP} An Error record might be a proxy error record. See System RAS Agents.

R_{PGPVB} An Error record might be a continuation record. See Multiple error records per node.

2.4.2 Component error states

R_{VWSSX} When a node records an error, the component error state is recorded in the Error record.

I_{KXNHF} The component error state recorded in the Error record describes the error state of the component only. For example, the component error state might be Unrecoverable state but the system is recoverable by resetting the component.

R_{LBBPN} For a standard Error record, the component error state types that can be recorded are:

- Corrected (CE).
- Deferred (DE).
- Uncorrected.

R_{KFPDF} If and only if all of the following are true, then on recording an error, the component error state is recorded as *Corrected* (CE):

- The error was corrected.
- The error has not been silently propagated.
- The component has not entered a service failure mode and continues to operate.
- The implementation has not elected to record the component error state as Deferred or Uncorrected.

In normal circumstances, the error no longer infects the state of the component. However, in the case of a persistent correctable fault, or other rare IMPLEMENTATION DEFINED circumstances, the error might remain latent in the component.

R_{XJFMG} If and only if all of the following are true, then on recording an error, the component error state is recorded as *Deferred* (DE):

- At least one of the following are true:
  - The error was not corrected, and was deferred.
  - The error was corrected, and the implementation elected to record the component error state as 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 component or whether it has been deferred to a consumer.
- The component has not entered a service failure mode and continues to operate.
- The implementation has not elected to record the component as Uncorrected.

I_{RFZHC} The component error state might be recorded as Deferred for an error that cannot be corrected. However, for the purposes of the component error state taxonomy, Deferred is classified separately from Uncorrected.

R_{KJTQQ} If and only if all of the following are true, then on recording an error, the component error state is recorded as *Uncorrected*:

- At least one of the following are true:
  - The error was not corrected and not deferred.
  - The error might have been silently propagated.
  - The component has entered as service failure mode and does not continue to operate the function that consumed the error.
  - The error was either corrected or deferred, and the implementation elected to record the component error state as Uncorrected.
- The error is latent in the system.

R_{WHGSP} When the component error state is recorded as Uncorrected, the node also records a sub-type in the Error record. The sub-types that can be recorded are:

- Uncontainable (UC).
- Unrecoverable state (UEU).
- Recoverable state or Signaled error (UER).
- Restartable state or Latent error (UEO).

R_{PHLQQ} If any of the following are true, then on recording the component error state as Uncorrected, the sub-type is recorded as *Uncontainable* (UC):

- The error might have been silently propagated by the component.
- The implementation has elected to record the component error state as Uncontainable.

If the error cannot be isolated, then the system must be shut down to avoid catastrophic failure.

R_{CTYHC} If and only if all of the following are true, then on recording the component error state as Uncorrected, the sub-type is recorded as *Unrecoverable state* (UEU):

- The error has not been silently propagated by the component.
- Either of the following are true:
  - The component has halted operation (entered a service failure mode) of the function that consumed the error. The component determines that software will not be able to recover operation of the function.
  - The implementation has elected to record the component error state as Unrecoverable state.

- The implementation has not elected to record the component error state as Uncontainable.

I_{DCYVN} As described by R_{WHGSP}, for an uncorrected error, the Error record records the component error state as one of UC, UEU, UER, or UEO. UER and UEO have two possible interpretations:

- Recoverable state and Restartable state, respectively.
- Signaled error and Latent error, respectively.

For each Error record, it is IMPLEMENTATION DEFINED which pair of meanings UER and UEO have. This might depend on the type of component:

- Signaled error and Latent error are more applicable to a *producer* or *Completer* component. For example, one that stores or transports data, such as memory or a cache.
- Recoverable state and Restartable state are more applicable to a *consumer* or *Requester* component. For example, one that might consume data and performs some operation on it.

R_{CNBRY} When UER means a Signaled error, if and only if all of the following are true, then on recording the component error state as Uncorrected, the sub-type is recorded as *Signaled error* (UER):

- The error was produced at the component.
- The error has not been silently propagated by the component.
- The error has been or might have been consumed, and the component error state was not recorded as a Deferred.
- The implementation has not elected to record the component error state as Unrecoverable state or Uncontainable.

R_{FFTXZ} When UEO means a Latent error, if and only if all of the following are true, then on recording the component error state as Uncorrected, the sub-type is recorded as *Latent error* (UEO):

- The error was produced at the component.
- The error has not been propagated by the component, silently or otherwise.
- The implementation has not elected to record the component error state as Deferred, Unrecoverable state, or Uncontainable.

That is, the error was detected but not consumed, and the component error state was not recorded as Deferred.

I_{NGDWZ} The producer is usually unable to determine whether a consumer has architecturally consumed the error. An error might be recorded as Latent error if it has definitely not been propagated to any consumer, and as Signaled error otherwise.

R_{QTYFD} When UER means a Recoverable error, if and only if all of the following are true, then on recording the component error state as Uncorrected, the sub-type is recorded as *Recoverable state* (UER):

- The error has not been silently propagated by the component.
- The component has halted operation (entered a service failure mode) of the function that consumed the error.
- Either of the following is true:
  - The component is reliant on consuming the corrupted data to continue operation of the function that consumed the error. The component determines that software will be able to recover operation of the function if it locates and repairs the error.
  - The implementation has elected to record the component error state as Recoverable state.
- The implementation has not elected to record the component error state as Deferred, Unrecoverable state, or Uncontainable.

R_{CFZTH} When UEO means a Restartable error, if and only if all of the following are true, then on recording the component error state as Uncorrected, the component error state is recorded as *Restartable state* (UEO):

- The error has not been silently propagated by the component.
- The component has halted operation (entered a service failure mode) of the function that consumed the error.
- The component determines that it does not rely on the corrupted data, and so can recover operation even if software does not locate and repair the error.
- The implementation has not elected to record the the component error state as Deferred, Unrecoverable state, or Uncontainable.

I_{TVJNM} The component error state types are summarized by Figure 2-1. Figure 2-1 assumes the component supports the resulting component error state and the implementation never elects to record an error as a different component error state when permitted.

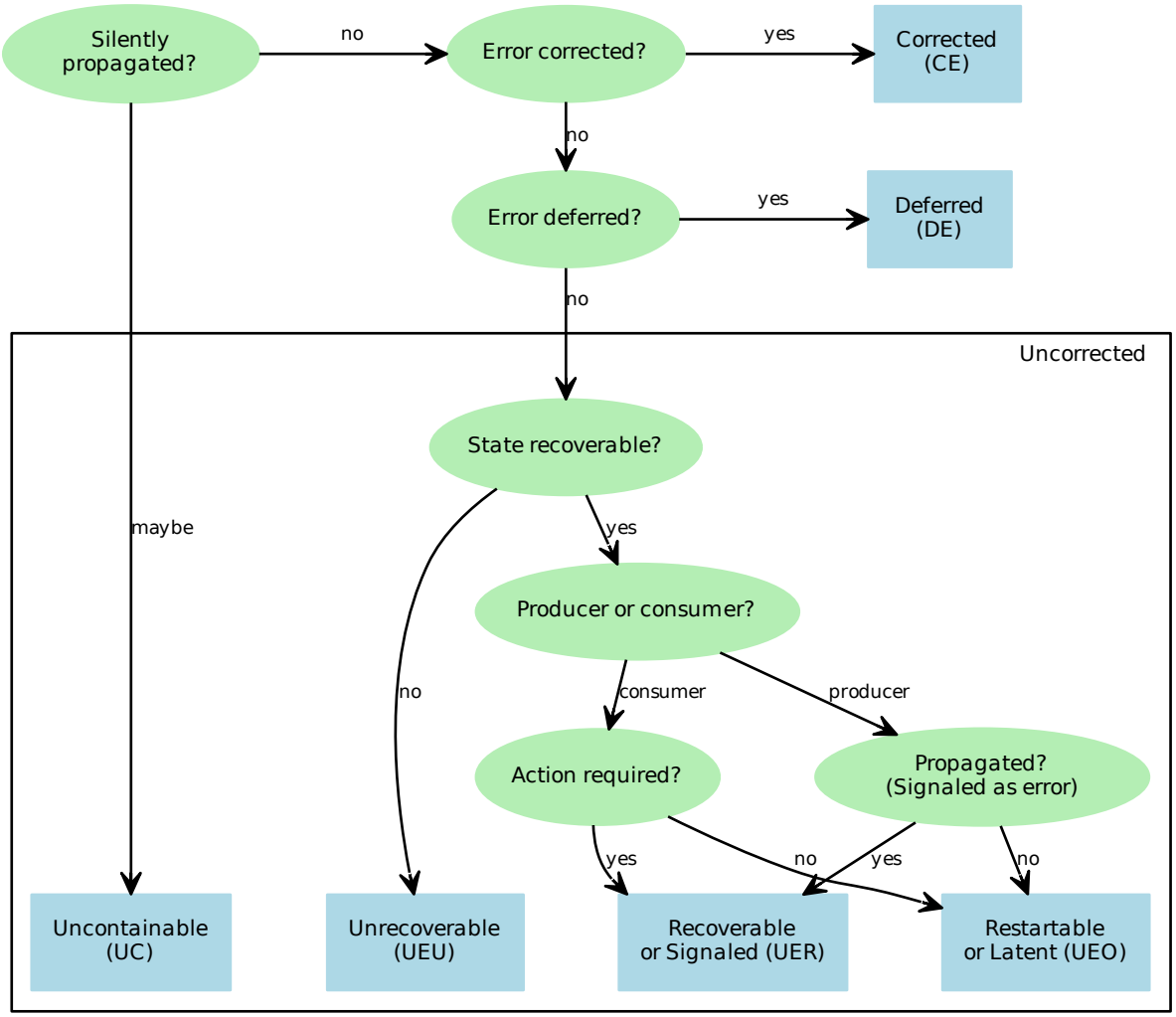


Figure 2-1 Component error state types

2.4.3 Writing the error record

R_{MDXXV} When a new error is recorded, the node:

- Does one of the following:
  - Overwrites the Error record with the syndrome for the new error.
  - Keeps the syndrome for the previous error.

  The previous component error state and the new component error state determine which. See:
  - Prioritizing errors, FEAT_RASSAv1.
  - Prioritizing errors, FEAT_RASSAv1p1.
- Modifies ERR<n>STATUS.{CE, DE, UE} to indicate the component error state. See Component error states and priorities.
- Counts the error, if a Corrected error counter is implemented and the error is of a type that the counter counts.

R_{TQKFF} If the Error record is corrupt or the previous component error state is otherwise not known, the node overwrites the Error record with the new error syndrome and sets ERR<n>STATUS.OF to `0b1`.

R_{BGXQQ} If counting a deferred error 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 Corrected error counter overflow. However, if the RAS System Architecture requires that recording the deferred error or uncorrected error sets the ERR<n>STATUS.OF flag to `0b1`, then this flag is also set to `0b1` even if the error is counted and the Corrected error counter does not overflow.

2.4.3.1 Component error states and priorities

R_{PXCDZ} The highest priority recorded component error state type is recorded in the ERR<n>STATUS.{V, CE, DE, UE, UET} fields, as shown in Table 2-1.
In Table 2-1, V, CE, DE, UE, UET refer to fields in ERR<n>STATUS.

Table 2-1 Encoding the highest priority component error state

ERR<n>STATUS					Highest priority component error state type	Mnemonic
V	CE	DE	UE	UET		
0b0	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	None (not valid)	-
0b1	0b00	0b0	0b0	UNKNOWN	None	-
0b1	!= 0b00	0b0	0b0	UNKNOWN	Corrected	CE
0b1	X	0b1	0b0	UNKNOWN	Deferred	DE
0b1	X	X	0b1	0b10	Uncorrected: Latent error or Restartable state	UEO
0b1	X	X	0b1	0b11	Uncorrected: Signaled error or Recoverable state	UER
0b1	X	X	0b1	0b01	Uncorrected: Unrecoverable state	UEU
0b1	X	X	0b1	0b00	Uncorrected: Uncontainable	UC

I_{QHBGV} The component error state types implemented at a node are IMPLEMENTATION DEFINED. An implementation might only include a simplified subset of these component error state types.
A node can always elect to record:

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

2.4.3.2 Prioritizing errors, FEAT_RASSAv1

R_{ZPTXT} When FEAT_RASSAv1p1 is not implemented, overwriting depends on the component error state type of the previous highest priority error and on the component error state type of the newly recorded error, as shown in Table 2-2.
In Table 2-2:

- Each row corresponds to the highest priority previous component error state type recorded in the Error record.
- Each column corresponds to the component error state type of the new detected error.

The row and column headings use the mnemonics from Table 2-1, 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 0b1 or unchanged.

O

Overflow. Keep the previous error syndrome and set ERR<n>STATUS.OF to 0b1.

W

Overwrite. Overwrite with the new error syndrome. It is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0b0 or unchanged.

CK

Count and keep. Count the error if a Corrected error counter is implemented, and keep the previous error syndrome. If the counter overflows, or if no Corrected error counter is implemented, then it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0b1 or unchanged.

CWK

Count and overwrite or keep. The behavior is IMPLEMENTATION DEFINED and described by the value of ERR<q>FR.CEO, where <q> is the index of the first Error record owned by the node:

- 0b00: Count the error if a Corrected error counter is implemented. Keep the previous error syndrome.

- `0b01`: Count the error. If ERR<n>STATUS.OF is `0b1` before the error is counted, then keep the previous syndrome. Otherwise, overwrite with the new error syndrome.

If counting the error causes unsigned overflow of the counter, or if no Corrected error counter is implemented, then ERR<n>STATUS.OF is set to `0b1`.

CW

Count and overwrite. Count the error if a Corrected error counter is implemented, and overwrite with the new error syndrome. If a Corrected error counter is implemented and counting the error causes unsigned overflow of the counter, then ERR<n>STATUS.OF is set to an UNKNOWN value. Otherwise, it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to `0b0` or unchanged.

WO

Overwrite and overflow. Overwrite with the new error syndrome. ERR<n>STATUS.OF is set to `0b1`.

Table 2-2 FEAT_RASSAv1 rules for overwriting error records

Previous error type	New detected errror 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

2.4.3.3 Prioritizing errors, FEAT_RASSAv1p1

R_{PNFPB}

When FEAT_RASSAv1p1 is implemented, overwriting depends on the component error state type of the previous highest priority error and on the component error state type of the newly recorded error, as shown in Table 2-3.
In Table 2-3:

- Each row corresponds to the highest priority previous component error state type recorded in the Error record.
- Each column corresponds to the component error state type of the new detected error.

The row and column headings use the mnemonics from Table 2-1, and the following additional abbreviations are used:

W

Overwrite. Overwrite with the new error syndrome. ERR<n>STATUS.OF is unchanged.

WO

Overwrite and overflow. Overwrite with the new error syndrome. ERR<n>STATUS.OF is set to `0b1`.

O

Overflow. Keep the previous error syndrome and set ERR<n>STATUS.OF to `0b1`.

If no Corrected error counter is implemented, then all of the following apply:

CW

Behaves the same as W.

CWO and CO

Behave the same as O.

Otherwise, a Corrected error counter is implemented, and all of the following apply:

CW

Count and overwrite. Overwrite with the new error syndrome, and count the error. If counting the error causes unsigned overflow of the counter, then ERR<n>STATUS.OF is set to `0b1`.

CWO

Count, overwrite or keep, and overflow. The behavior is IMPLEMENTATION DEFINED and described by the value of ERR<q>FR.CEO, where <q> is the index of the first Error record owned by the node:

- `0b00`: The behavior is the same as CO.
- `0b01`: Count the error. If ERR<n>STATUS.OF is `0b1` before the error is counted, then the behavior is the same as CO. Otherwise, the behavior is the same as CW.

CO

Count and overflow. Keep the previous error syndrome, and count the error. If counting the error causes unsigned overflow of the counter, then ERR<n>STATUS.OF is set to `0b1`.

Table 2-3 FEAT_RASSAv1p1 rules for overwriting error records

Previous error type	New detected errror type					
	CE	**DE**	**UEO**	**UER**	**UEU**	**UC**
-	CW	W	W	W	W	W
CE	CWO	WO	WO	WO	WO	WO
DE	CO	O	WO	WO	WO	WO
UEO	CO	O	O	WO	WO	WO
UER	CO	O	O	O	WO	WO
UEU	CO	O	O	O	O	WO
UC	CO	O	O	O	O	O

2.4.3.4 Overwriting the error syndrome

R_{RVGRM} When the node records an error in an Error record and either the previous syndrome is overwritten with the new error syndrome, or the Error record was previously not valid:

- Modifies ERR<n>STATUS.{V, CE, DE, UE} to indicate the new component error state, as described by Table 2-1:
  - Fields shown as `x` in Table 2-1 are unchanged.
  - Other ERR<n>STATUS.{V, CE, DE, UE} fields are set to the value given in Table 2-1.

  If the new component error state is Corrected, then the nonzero value written to ERR<n>STATUS.CE is IMPLEMENTATION DEFINED and depends on the properties of the corrected error recorded.
- If the new component error state is Uncorrected, then ERR<n>STATUS.UET is set to indicate the component error state sub-type. See Component error states and priorities.
- 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, then ERR<n>STATUS.AV is set to `0b1` and the address is written to ERR<n>ADDR. Otherwise ERR<n>STATUS.AV is set to `0b0` and ERR<n>ADDR becomes UNKNOWN.
- If the RAS Timestamp Extension is implemented, then a timestamp is recorded in ERR<n>MISC3 and ERR<n>STATUS.MV is set to `0b1`.
- If there is other miscellaneous syndrome for the new error, then the syndrome is written to the ERR<n>MISC<m> registers and ERR<n>STATUS.MV is set to `0b1`.
- If there is no additional miscellaneous syndrome for the new error written to the ERR<n>MISC<m> registers, then it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.MV is set to `0b0` or unchanged.
  - If software can determine from the ERR<n>MISC<m> contents that the syndrome is not related to the highest priority error, then the ERR<n>STATUS.MV bit is unchanged.
  - Otherwise the ERR<n>STATUS.MV bit is cleared to zero.
- ERR<n>STATUS.V is set to `0b1`.

S_{XFYQK} After reading an ERR<n>STATUS register, software has to write to the register to clear the valid bits to 0 in the register to allow new errors to be recorded. During this period, a new error might overwrite the syndrome for the previously read error. To prevent this, the write, or part of the write, is ignored by hardware if fields appear to have been updated. For more information see ERR<n>STATUS.

2.4.3.5 Keeping the previous error syndrome

R_{BGBBD} When the previous Error record is kept:

- Sets the applicable one of ERR<n>STATUS.{CE, DE, UE} to indicate the new component error state:
  - If Uncorrected, then ERR<n>STATUS.UE is set to `0b1`.
  - If Deferred, then ERR<n>STATUS.DE is set to `0b1`.
  - If Corrected, then the nonzero value written to ERR<n>STATUS.CE is IMPLEMENTATION DEFINED and depends on the properties of the corrected error recorded.

  The remaining ERR<n>STATUS.{UE, DE, CE} fields are unchanged.
- ERR<n>STATUS.UET is unchanged, even if the new component error state is Uncorrected.
- ERR<n>STATUS.{ER, PN, IERR, SERR}, ERR<n>ADDR, and ERR<n>STATUS.AV are unchanged.
- If the RAS Timestamp Extension is implemented, then the timestamp is not recorded.
- It is IMPLEMENTATION DEFINED whether any of ERR<n>MISC<m> are updated. The contents of ERR<n>MISC<m> 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 any of ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to `0b1`.

2.4.3.6 Detecting multiple errors

R_{RXQWW} If multiple errors are simultaneously reported to a node, then 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.

This includes any error reported by a fault injection mechanism. If one of the multiple errors is due to a fault injection mechanism, then it is IMPLEMENTATION SPECIFIC whether the behavior described by R_{XLJMM} applies.

R_{ZJXMD} If a Corrected error counter is implemented, and multiple countable errors are detected simultaneously, then:

- If at least one of the errors is countable, then at least one of the errors is counted.
- Otherwise, it is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether any other of the errors are counted.

I_{WNKMD} If a pair of error counters that count *repeat* and *other* errors are implemented, and the multiple countable errors comprise at least one *repeat* error and at least one *other* error, then Arm recommends that at least one repeat error and at least one other error are counted. R_{XYFVB} and I_{FYBWQ} describe such an implementation.

I_{PHBRX} See also Standard format Corrected error counter.

2.4.4 Error syndrome

I_{YLHWP} This section provides additional information for some of the error syndrome fields defined in the standard Error record.

2.4.4.1 Corrected error field

I_{BRMPK} When a syndrome with a component error state of Corrected is recorded, the node also records one of the following in the ERR<n>STATUS.CE error type field:

- The component or node has determined that the error is transient, or likely to be so.
- The component or node has determined that the error is persistent, or likely to be so.
- The component or node does not support making such a determination or is unable to.

R_{FCQDJ} The mechanism by which a component or node determines whether a corrected error is transient or persistent is IMPLEMENTATION DEFINED.

2.4.4.2 Poison indicator

I_{DTYHM} If supported by a node, then when a syndrome with a component error state of Deferred or Uncorrected is recorded, the ERR<n>STATUS.PN syndrome field is set to indicate that a poisoned value was detected.

R_{PNKSH} When the node records an error and overwrites the previous error syndrome, if all of the following are true the ERR<n>STATUS.PN syndrome field is set to `0b1`, and is set to `0b0` otherwise:

- The component checks a value for an error and detects the value indicates a previously deferred error. For example, the value is a poisoned value.
- The node does one of the following:
  - Records the component error state as Uncorrected. For example, because the component does one or more of:
    - Enters a service failure mode.
    - Propagates the value to a component that does not support poison. That is, the error has been silently propagated.
  - If the component has deferred the error again, records the component error state as Deferred. See also Bridges to other architectures.

I_{JBDPT} When a component checks a value and detects an uncorrectable error, and defers the error by generating a poisoned value, the node records the component error state as Deferred with ERR<n>STATUS.PN set to `0b0`.

Therefore when software examines the Error records, an ERR<n>STATUS.PN value of `0b1` indicates that the component was propagating a previously deferred error, and so the fault did not originate in that component. An ERR<n>STATUS.PN value of `0b0` indicates that the fault originated at the component.

I_{QLSMY} In some *Error Detection Code* (EDC) schemes, a poisoned value is encoded as a reserved value, one that would not be generated by a detectable corruption of valid data.

Example 2-4 Encoding poisoned values

In a SECDED error detection scheme, a value with a Hamming distance greater than 2 bits from all valid values is chosen to represent a poisoned value.

For such a scheme, it is IMPLEMENTATION DEFINED whether the component can distinguish a corrupt data value from the poison value. The component might accept and store a poisoned value when an error is deferred to it, but treat it as any other uncorrectable error when it is accessed, meaning ERR<n>STATUS.PN is set to `0b0`.

2.4.5 Security and Virtualization

2.4.5.1 Confidential data

I_{CLQTQ} **In a system with FEAT_RME:**

In normal operation, when a Security state cannot access data because that data is from a different Security state, that data is *confidential data*. For example:

- Non-secure state cannot access data from any other Security state. When executing in Non-secure state, data from all other Security states is confidential.

When accessed from a PE executing in:

- Realm state, data from Secure and Root states is confidential.
- Secure state, data from Realm and Root states is confidential.
- Root state, there is no confidential data.

Confidential data comprises all of:

- Confidential data in memory locations, including locations that the *Granule Protection Table* (GPT) prohibits access to.
- Confidential data in registers: SIMD&FP, SVE, SME, System, Special-purpose, and general-purpose registers.

All the following are considered to be *always non-confidential* data:

- Addresses at which errors are detected, captured in ERR<n>ADDR registers.

Note

There are exceptions in the case of error injection. See The Common Fault Injection Model Extension.

- Identities of FRUs, captured in ERR<n>STATUS and/or ERR<n>MISC<m> registers.
- Information about the severity of an error, such as:
  - Error record status information captured in ERR<n>STATUS registers.
  - Error counters.
- Information used to ascertain the priorities of an error node, identification, and affinity.

I_{XLTYW} **In a system without FEAT_RME:**

Which data is categorized as *confidential data* is IMPLEMENTATION SPECIFIC and depends on how the information encoded in the data relates to the threat model for the system.

Example 2-5 Categorizing confidential data

Data from Secure state might be called *Secure data*. Non-secure state cannot access Secure data, therefore when executing in Non-secure state, Secure data is categorized as confidential data.

Confidential data comprises all of:

- Confidential data in memory locations.
- Confidential data in registers: SIMD&FP, SVE, SME, System, Special-purpose, and general-purpose registers.

I_{TLCJT} The *highest Security state* is:

- Root state if FEAT_RME is implemented.
- Secure state otherwise.

R_{SXKNQ} Error detection and correction for accesses to memory assigned to the:

- Secure physical address space, cannot be disabled by either:
  - Controls accessible in the Non-secure or Realm physical address spaces.
  - A PE executing in Non-secure or Realm state.
- Realm physical address space, cannot be disabled by either:
  - Controls accessible in the Non-secure or Secure physical address spaces.
  - A PE executing in Non-secure or Secure state.
- Root physical address space, cannot be disabled by any of:
  - Controls accessible in the Non-secure, Secure, or Realm physical address spaces.
  - A PE executing in Non-secure, Secure, or Realm state.

I_{PWXNT} Arm strongly recommends that all the following apply:

- Error detection and correction for accesses to shared resources, and for memory that can be assigned to any physical address space, cannot be disabled by controls accessible to all Security states.
- Any configuration that can control error detection and correction is writable in the highest Security state only, or firmware can block write access to the configuration by using a control that is writable in the highest Security state only.

R_{DDQTB} In a system with FEAT_RME, error signaling and recording controls for error records that might contain confidential data are accessible to a PE executing in Root state only.

I_{VKZNJ} In a system without FEAT_RME, Arm recommends that error signaling and recording controls for error records that might contain confidential data are accessible to a PE executing in Secure state only.

I_{FMRZD} Memory contents that are encrypted without freshness are considered as confidential as their corresponding plaintext.

R_{SKYHW} Scrubbers and DMAs must report faults or errors back to an agent that can attribute the error back to the owner.

2.4.5.2 Security of error records

R_{VGSMG} For memory-mapped components, accesses to error records from:

- Non-secure state do not expose Secure, Realm, or Root data.
- Secure state do not expose Realm or Root data.
- Realm state do not expose Secure or Root data.

This might be guaranteed by the implementation, or by software executing at the highest Security state, or both.

I_{TFGNP} To achieve R_{VGSMG}, a number of implementation options are possible, for example:

- Error records contain always non-confidential data only.
- The Security state accessing the error record defines the data that the error record exposes. For example, for an access from Secure state:
  - Data from Realm and Root states is confidential. The error record cannot expose this.
  - Data from Secure and Non-secure states is non-confidential. The error record can expose this.
- Error records that might contain confidential data are accessible to the highest Security state only:
  - For memory-mapped components, they are accessible in the physical address space corresponding to the highest Security state only.
  - If a PE implements System register access to error records, software can use PE Trap exception controls to ensure that error records that might contain confidential data are accessible to the highest Security state only.
- FEAT_RASSA_ACR might be implemented.

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

- Error records are accessible to all Security states.
- Error records are accessible to the highest Security state only.
- The ERRACR register is implemented.

For each Security state, it can be configurable whether error records are accessible.

Arm strongly recommends against making all error records accessible to the highest Security state only.

I_{FCRRD} See also:

- Confidential data.

D_{VKJVV} FEAT_RASSA_ACR is an OPTIONAL Error record group feature from FEAT_RASSAv1p1.

I_{MTYGN} When FEAT_RASSA_ACR is implemented, a trusted agent is able to restrict control over error detection, correction, and signaling for data owned by a Security state to at least the owner of the data or the trusted agent, and prevent untrusted agents from being able to modify error records that might be used by trusted agents.

R_{FTLLR} When FEAT_RASSA_ACR is implemented by an Error record group, the group includes the *Access Control Register*, ERRACR.

S_{NXXBP} FEAT_RASSA_ACR is identified to software by ERRACR.IMPL.

R_{QZWVX} When FEAT_RASSA_ACR is implemented and FEAT_RME is not implemented, all of the following apply:

- Each Error record group has views in the Secure and Non-secure physical address spaces (PASs).
- The Secure PAS view includes ERRACR.
- ERRACR controls Non-secure access to the error records.
- ERRACR is RAZ/WI in the Non-secure PAS view.

R_{VSVLC} When FEAT_RASSA_ACR is implemented and FEAT_RME is implemented, all of the following apply:

- Each Error record group has views in the Root, Secure, Realm, and Non-secure physical address spaces (PASs).
- The Root PAS view includes ERRACR.
- ERRACR controls Secure, Realm, and Non-secure access to the error records.
  It is IMPLEMENTATION DEFINED whether ERRACR includes one control that applies to all PASs other than Root, or one control per other PAS.
- ERRACR is RAZ/WI in the Secure, Realm, and Non-secure PAS views.

S_{HSDRT} Providing controls per PAS allows system firmware that trusts Secure state software to handle RAS configuration, but not Realm or Non-secure states.

R_{QSFYN} The access control levels provided for a PAS in ERRACR are:

- Access is disabled. All registers are RAZ/WI.
- Read-only access is enabled. All error record and interrupt configuration registers ignore writes. The effect on accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.
- Read/write access is enabled.

Access to error records from reset is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

I_{KFNCK} The *read-only* access level applies to all error record registers (ERR<n>*, including, if implemented, the fault injection registers ERR<n>PFG*), and interrupt configuration registers (ERR<irq>CR<m> and, if implemented, ERRIRQSR) in the Error record group.

R_{VKJDY} When FEAT_RASSA_ACR is implemented and ERRACR allows Non-secure or Realm writes to an ERR<irq>CR<m> register, ERR<irq>CR2.NSMSI is ignored and writes for that message signaled interrupt are always to the Non-secure PAS.

I_{XBPWR} ERRACR is intended for use by system firmware. An implementation might include IMPLEMENTATION DEFINED equivalent controls in a different location, for example, outside of the Error record group page.

I_{FCDDX} An implementation might extend the ERRACR to control other aspects of the Error record group.

For example, by default the access PAS also determines what *confidential information* is visible in other views, when FEAT_RME is implemented. However, an implementation might include *confidential information* observability controls for different PAS accesses.

For example, fine-grained write access controls might be defined for the ERR<n>CTLR and ERR<irq>CR<m> registers:

- Per-interrupt write access controls for control fields relating to each of the FHI, ERI, and CRI interrupts.
- Write access control for the *in-band error response* control, ERR<n>CTLR.UE.
- Write access control for the *error detection* enable control, ERR<n>CTLR.ED.

I_{LFCBW} Common Fault Injection Model Extension registers can be placed in a separate fault injection group page, with ERR<n>PFGCTL for node *n* located at `0x000`+64×*n*.

2.4.5.3 IMPLEMENTATION DEFINED fault or error injection models

R_{ZBSKS} If FEAT_RME is implemented:

- IMPLEMENTATION DEFINED error injection mechanisms must not corrupt any data.
- IMPLEMENTATION DEFINED error injection models that support signaling an error on a PE accessing a specific physical address must not be implemented when all the following are true:
  - The address can be set by PEs not in Root state.
  - The error injection controls are accessible to PEs not in Root state.

R_{NYGDN} If FEAT_RME is not implemented, Arm recommends:

- IMPLEMENTATION DEFINED error injection mechanisms do not corrupt any data.
- Not to allow all of the following if the error injection models that support signaling an error on a PE accessing a specific physical address:
  - The address can be set by PEs not in Secure state.
  - The error injection controls are accessible to PEs not in Secure state.

2.4.6 Synchronization and error record accesses

R_{VKYVJ} When a component reports an error to a node, the node updates the Error record registers and might generate one or more of the following:

- A Fault handling interrupt.
- An Error recovery interrupt.

- A Critical Error interrupt.
- An In-band error response.

Each of these might generate an exception at a PE.

R_{VYCRY} 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 returns the updated values. This applies for both:

- Error records accessed through memory-mapped registers, only if the memory-mapped registers are mapped as a Device type that does not permit read speculation.
- Error records accessed through System registers, only if either the exception is a Context synchronization event or a Context synchronization event occurs in program order after taking the exception and before reading the System registers

R_{NHZBG} When a component reports an error to node, the node updates the Error record registers in finite time, and the update is globally observed for all observers in the system in finite time.

2.4.7 Bridges to other architectures

R_{LWGCK} A bridge is a component that passes transactions between two domains. For example, a bridge between an SoC domain and a *Peripheral Component Interconnect Express* (PCIe) domain.

I_{FKKVY} As described in Error propagation, a high-level transaction might consist of a sequence of operations passed between the domains by the bridge. For the purposes of this manual, the most basic form of a unidirectional transfer between a *producer* and *consumer* is considered as a transaction. That is, each one of the sequence of operations is a transaction.

R_{ZXBSX} Other standards might define mechanisms for RAS error recording and handling in particular domains.

I_{YQMVB} In the case of PCIe, the PCIe domain might implement one or more of:

- Simple error recording. Errors are recorded in the PCIe device status register.
- PCIe *advanced error reporting* (AER). Errors are recorded in the AER logs.
- Vendor-specific error recording. Errors are recorded in *Designated-Vendor-Specific Extended Capability* (DVSEC) logs.

In each case, errors detected in the PCIe domain are recorded in the PCIe domain and not in the SoC domain.

I_{YTXWG} For the purposes of tracking the origins of a detected error or a deferred error that has propagated between domains, it may be useful to record when a transaction propagates a detected error or a deferred error to a different domain.

Arm recommends that a bridge between domains, where the domains implement different error recording mechanisms, uses a node to record when a transaction that is signaled as propagating either a detected error or a deferred error crosses between the domains, recording the source and direction of the transaction in the IMPLEMENTATION DEFINED syndrome for the Error record. The direction is either *inbound* or *outbound*.

2.4.8 Software faults

I_{SSQXP} Examples of software faults include:

- Access to memory or peripheral register that is not present. This includes cases where physical address spaces are physically aliased.
- Access to a peripheral that is not permitted at the completer. For example, a Non-secure access to a Secure register.
- Access to a peripheral that is in an inaccessible state or other illegal access. For example, the peripheral is powered down, or the value written is not supported.

I_{BYWQQ} Software fault handling is outside the scope of the RAS System Architecture. Arm makes the following recommendations for accesses that constitute a software fault:

- Accesses to a memory location that is not present can return an In-band error response when all of the following are true:

  - The location is *not present* due to a configuration of the physical address map that is either static or controlled by trusted software. For example, a configuration choice made by the designer, set during initial system configuration, or reconfigured by trusted software.
    It is not because a peripheral has been unexpectedly removed or the address map has been otherwise reconfigured. For example, when a user unplugs a peripheral, or using software controls intended to be available to untrusted software. The split between *trusted* and *untrusted* is IMPLEMENTATION SPECIFIC, but, for example untrusted would typically include unprivileged software and, in systems that supports virtualization, guest operating systems. *Untrusted* might or might not include Non-secure hypervisors.
  - Within the aligned page that contains the not-present location, all other locations are also *not present* and have the same behavior. The size of this page is the largest supported translation granule size of all PEs in the system.

  That is, there is never any legitimate reason for software to access the page containing the location, and trusted software should set up the translation tables to prevent accesses from occurring.
- Where another standard defines a rule or sets a convention, that should be followed. For example:
  - For a PCIe device, certain illegal accesses are RAO/WI or can have their behavior configured by software.
  - The Arm architecture requires that *reserved accesses* to a component behave as RAZ/WI. This includes reads and writes of unallocated or unimplemented registers and writes to read-only registers.
  - The Arm architecture requires that under certain conditions accesses to certain debug registers return an error response.

For other cases, the access should do one of the following:

- Return zeros to the requester for a read and ignore writes. This is the recommended behavior for reads and writes of unallocated or unimplemented registers, reads of write-only registers, and writes of read-only registers.
- Return all-ones to the requester for a read and ignore writes.
- Return an IMPLEMENTATION DEFINED value to the requester for a read and ignore writes.

In some implementations, this is done by the completer of the access.

In other implementations, this might be done by a bridge wrapper for a component or components that do not natively support recording a software fault. The wrapper detects and suppresses an In-band error response from the completer and responds to the requester appropriately. Such a wrapper might be configurable and might also record the software fault, as described by I_{NXCDR}.

If the system does not support any means to record the software fault, then an In-band error response should not be returned to the requester.

I_{NXCDR} The system might implement a RAS System Architecture node or nodes and Error records to record software faults, for improved debuggability of the faults.

When a node and Error records for recording software faults is implemented, software faults can be recorded as an error, and reported with an In-band error response and/or a Fault handling interrupt, referred to as a software fault interrupt. Arm recommends that this is configurable through ERR<n>CTLR, allowing software to disable the feature. For example, if an Error exception might cause an unrecoverable software state.

When the feature is disabled, accesses should behave as recommended above.

The following ERR<n>STATUS.SERR values can be used to record software faults.

SERR	Description
0x0D	Illegal address (software fault). For example, access to unpopulated memory.
0x0E	Illegal access (software fault). For example, byte write to word register.
0x0F	Illegal state (software fault). For example, device not ready.
0x19	Error recorded by PCIe error logs. Indicates that the node hasrecorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.

2.4.9 Other sources of error and warnings

I_{NWXQS} Other sources of error and warning are possible in a system. Within the RAS System Architecture, these are signaled to a PE using an Error recovery interrupt or Fault handling interrupt.

2.5 RAS interrupts

2.5.1 Overview

I_{BHBCB} Error recovery, Fault handling, and Critical Error interrupt requests are normally routed to a PE using an interrupt controller.

I_{QTQBJ} For an Arm *Generic Interrupt Controller* (GIC), if the Error records of the node that generates the interrupt requests are only accessible via the System registers of one or more PEs, Arm strongly recommends that the interrupt is a *Private Peripheral Interrupt* (PPI) targeting that PE or one of those PEs.

R_{VKLWD} It is IMPLEMENTATION DEFINED whether each Error record has independent interrupt request signals for Error recovery, Fault handling, and Critical Error interrupt requests, or whether it shares any of these interrupt requests with other Error records and/or other nodes.

R_{WMQZP} It is IMPLEMENTATION DEFINED whether interrupt requests are edge-triggered or level-sensitive.

R_{BRKDL} It is IMPLEMENTATION DEFINED whether interrupt requests are implemented as a direct connection (wire) to an interrupt controller or controllers, as an *Message Signaled Interrupt* (MSI), or both.

R_{SVWPZ} The Fault handling condition for an Error record <n> is true if and only if any of the following apply:

- Fault handling interrupts on all uncorrected errors are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.UE is `0b1`.
- Fault handling interrupts on all deferred errors are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.DE is `0b1`.
- Fault handling interrupts on corrected errors are enabled and either:
  - The Error record implements a Corrected error counter, ERR<n>STATUS.V is `0b1`, and any of the following apply:
    - The counter overflow flag is `0b1`, and either FEAT_RASSA_CED is not implemented or ERR<n>CTLR.CED is `0b0`.
    - FEAT_RASSA_CED is implemented, ERR<n>CTLR.CED is `0b1`, and ERR<n>STATUS.CE is nonzero.
  - The Error record does not implement a Corrected error counter, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.CE is nonzero.

R_{MJYYL} If the Fault handling interrupt for one or more Error records is level-sensitive, then:

- The Fault handling interrupt is asserted if the Fault handling condition is true for any of the Error records.
- The Fault handling interrupt is deasserted if the Fault handling condition is false for all of the Error records.

R_{VHSRJ} The Error recovery condition for an Error record <n> is true if and only if any of the following apply:

- Error recovery interrupts on uncorrected errors are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.UE is `0b1`.
- Error recovery interrupts on deferred errors are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.DE is `0b1`.

R_{ZJRQN} If the Error recovery interrupt for one or more Error records is level-sensitive, then:

- The Error recovery interrupt is asserted if the Error recovery condition is true for any of the Error records.
- The Error recovery interrupt is deasserted if the Error recovery condition is false for all of the Error records.

R_{KTVHF} The Critical Error condition for an Error record <n> is true if and only if Critical Error interrupts are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.CI is `0b1`.

R_{KMGJJ} If the Critical Error interrupt for one or more Error records is level-sensitive, then:

- The Critical Error interrupt is asserted if the Critical Error condition is true for any of the Error records.
- The Critical Error interrupt is deasserted if the Critical Error condition is false for all of the Error records.

R_{YPPWB} If the Fault handling interrupt is edge-triggered, then the interrupt request is generated by the node for an Error record when any of the following occur:

- Fault handling interrupts on all deferred and uncorrected errors are enabled, and an error is recorded in the Error record with a component error state of either Deferred or Uncorrected.
- Fault handling interrupts on corrected errors are enabled and a Corrected error event occurs for the Error record.

R_{FLWGK} If the Error recovery interrupt is edge-triggered, then the interrupt request is generated by the node for an Error record when any of the following occur:

- Error recovery interrupts on uncorrected errors are enabled, and an error is recorded in the Error record with a component error state of Uncorrected.
- Error recovery interrupts on deferred errors are enabled, and an error is recorded in the Error record with a component error state of Deferred.

R_{FLPKB} If the Critical Error interrupt is edge-triggered, then the interrupt request is generated by the node for an Error record <n> when Critical Error interrupts are enabled, and the node records an error setting ERR<n>STATUS.CI to `0b1`.

The Critical Error interrupt request is generated even if ERR<n>STATUS.CI was already `0b1`.

I_{MYKYF} An enabled edge-triggered interrupt request is generated even if the error syndrome is discarded because the Error record already records a higher priority error.

R_{XWMLB} It is IMPLEMENTATION DEFINED whether an edge-triggered interrupt request is generated by a write to a register that enables an interrupt or otherwise creates the conditions for the interrupt request in the other syndrome registers, as defined for a level-sensitive interrupt request.

D_{JWTHK} FEAT_RASSA_IRQCR_SIMPLE is an OPTIONAL Error record group feature from FEAT_RASSAv1.

D_{PRGRZ} FEAT_RASSA_IRQCR_MSI is an OPTIONAL Error record group feature from FEAT_RASSAv1.

S_{KMMSP} FEAT_RASSA_IRQCR_SIMPLE and FEAT_RASSA_IRQCR_MSI are identified to software by ERRDEVID.IRQCR from FEAT_RASSAv2.

The following are identified to software by ERRDEVID.IRQCR:

- Whether interrupt configuration registers are implemented.
- If interrupt configuration registers are implemented, whether the registers use the recommended format or are IMPLEMENTATION DEFINED.
- If the interrupt configuration registers use the recommended format, which of the following controls are implemented:
  - Simple interrupt controls, FEAT_RASSA_IRQCR_SIMPLE.
  - Message-signaled interrupt controls, FEAT_RASSA_IRQCR_MSI.

R_{GZQWV} The standard Error record reserves a set of register locations for configuring *Message Signaled Interrupts* (MSIs), ERRIRQCR<m>. Two recommended layouts for these registers are described by FEAT_RASSA_IRQCR_SIMPLE and FEAT_RASSA_IRQCR_MSI, as follows:

- If FEAT_RASSA_IRQCR_SIMPLE is implemented, an interrupt enable control for each of the Error recovery, Fault handling, and Critical Error interrupt requests is implemented. These are ERRERICR2.IRQEN, ERRFRICR2.IRQEN, and ERRCRICR2.IRQEN respectively. The reset value for these controls is IMPLEMENTATION DEFINED.
- If FEAT_RASSA_IRQCR_MSI is implemented, the Interrupt Status Register, ERRIRQSR, is implemented, and for each of the Error recovery, Fault handling, and Critical Error interrupt requests, three configuration registers are implemented:
  - Interrupt Configuration Register 0 holds the address to which the node writes to request the interrupt. These are ERRERICR0, ERRFHICR0, and ERRCRICR0 respectively.
  - Interrupt Configuration Register 1 holds the 32-bit data value that the node writes to the address. These are ERRERICR1, ERRFHICR1, and ERRCRICR1 respectively.
  - Interrupt Configuration Register 2 configures all the following:
    - Whether the message signaled interrupt is enabled or disabled.
    - The Shareability domain and memory type attributes for the address.
    - The physical address space for the address. This is either the Non-secure physical address space or the Secure physical address space.

    These controls and attributes are optional. These registers are ERRERICR2, ERRFHICR2, and ERRCRICR2 respectively.

If the recommended layouts are not used, then the ERRIRQCR<m> registers are IMPLEMENTATION DEFINED.

R_{RZDWL} When an error is recorded, or an interrupt becomes enabled, the state of the interrupt requests is updated in finite time.

2.5.2 Fault handling interrupt

I_{DZTCG} If a Fault handling interrupt is implemented by a node, then the set of controls for enabling Fault handling interrupts is IMPLEMENTATION DEFINED. Software uses ERR<n>FR to determine what controls are implemented.

D_{TSXMY} For a node <n>, ERR<n>CTLR.FI is the OPTIONAL control for generating the Fault handling interrupt.

D_{ZTFFN} FEAT_RASSA_DFI is an OPTIONAL node feature from FEAT_RASSAv1p1.

D_{DHJGJ} For a node <n>, ERR<n>CTLR.DFI is the OPTIONAL control for generating the Fault handling interrupt on deferred errors.

R_{NQWKT} For a node <n>, if FEAT_RASSA_DFI is implemented, then the ERR<n>CTLR.DFI control is implemented.

R_{QLBNC} For a node <n>, if FEAT_RASSA_DFI is implemented, then the ERR<n>CTLR.FI control is implemented.

D_{WHXMB} ERR<n>CTLR.CFI is the OPTIONAL control for generating the Fault handling interrupt on Corrected error events.

R_{TSPRZ} For a node <n>, if the ERR<n>CTLR.CFI control is implemented, then the ERR<n>CTLR.FI control is implemented.

R_{ZVPHD} For a node <n>, if the ERR<n>CTLR.FI control is implemented, then the Fault handling interrupt is enabled for uncorrected errors when ERR<n>CTLR.FI is `0b1`, and disabled for uncorrected errors when ERR<n>CTLR.FI is `0b0`.

R_{HSGLQ} For a node <n>, if FEAT_RASSA_DFI is implemented, then the Fault handling interrupt is enabled for deferred errors when ERR<n>CTLR.DFI is not equal to ERR<n>CTLR.FI, and disabled for deferred errors when ERR<n>CTLR.DFI is equal to ERR<n>CTLR.FI.

R_{JSKSW} For a node <n>, if the ERR<n>CTLR.FI control is implemented and FEAT_RASSA_DFI is not implemented, then the Fault handling interrupt is enabled for deferred errors when ERR<n>CTLR.FI is `0b1`, and disabled for deferred errors when ERR<n>CTLR.FI is `0b0`.

R_{QWFKB} For a node <n>, if the ERR<n>CTLR.CFI control is implemented, then the Fault handling interrupt is enabled for Corrected error events when ERR<n>CTLR.CFI is `0b1` and disabled for Corrected error events when ERR<n>CTLR.CFI is `0b0`.

R_{BSXNL} For a node <n>, if the ERR<n>FR.CFI control is not implemented, then the Fault handling interrupt is enabled for Corrected error events when the Fault handling interrupt for deferred errors is enabled, and disabled otherwise.

R_{MLJNK} For a node <n>, if the ERR<n>CTLR.FI control is not implemented, then the Fault handling interrupt is always enabled for all Corrected error events, deferred errors, and uncorrected errors.

R_{WFNLG} For a node <n>, if a Fault handling interrupt is not implemented, then the ERR<n>CTLR.{CFI,DFI,FI} controls are not implemented.

I_{XFWMB} When a node implements separate ERR<n>CTLR.{CFI,DFI,FI} controls, the node generates Fault handling interrupts as follows:

Table 2-5 Fault handling interrupt generation when FEAT_RASSA_DFI is implemented

ERR<n>CTLR			Fault handling interrupt generated by:		
CFI	DFI	FI	CE Event	DE	UE
0b0	0b0	0b0	No	No	No
0b0	0b0	0b1	No	Yes	Yes
0b0	0b1	0b0	No	Yes	No
0b0	0b1	0b1	No	No	Yes
0b1	0b0	0b0	Yes	No	No
0b1	0b0	0b1	Yes	Yes	Yes
0b1	0b1	0b0	Yes	Yes	No
0b1	0b1	0b1	Yes	No	Yes

D_{NDSNV} FEAT_RASSA_CED is an OPTIONAL node feature from FEAT_RASSAv1p1.

I_{WBBGW} FEAT_RASSA_CED allows software to switch to a mode where each corrected error generates a fault handling interrupt, without having to reset the error counter after each interrupt.

S_{FHBJT} FEAT_RASSA_CED is identified to software by ERR<n>FR.CED.

R_{CPLTL} When FEAT_RASSA_CED is implemented by a node, each error record <n> owned by the node that includes an error counter includes the *corrected error event from error counter disable* control, ERR<n>CTLR.CED.

R_{CGZMD} If the node implements a Corrected error counter, and either FEAT_RASSA_CED is not implemented or ERR<n>CTLR.CED is `0b0`, then all of the following are true:

- A Corrected error event occurs when a counter overflows and sets a counter overflow flag to `0b1`.
- It is UNPREDICTABLE whether a Corrected error event occurs when a software write sets the counter overflow flag to `0b1`.
- It is UNPREDICTABLE whether a Corrected error event occurs when a counter overflows and the overflow flag was previously set to `0b1`.

Otherwise, a Corrected error event occurs when the node records an error with a component error state of Corrected.

R_{JJHNT} If FEAT_RASSA_DFI is implemented and an error counter or counters are implemented, then the error counter counts deferred errors and corrected errors. It is IMPLEMENTATION DEFINED whether the counter counts uncorrected errors. This means that a deferred error might also cause a Corrected error event and, if the ERR<n>CTLR.CFI control is implemented, might generate a Fault handling interrupt when the Fault handling interrupt for deferred errors is disabled because the Fault handling interrupt for Corrected error events is enabled.

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

R_{DQWYH} The Fault handling interrupt is generated when the node records an error, even if the error syndrome is discarded because the Error record already records a higher priority error.

2.5.3 Error recovery interrupt

I_{JXHYH} If an Error recovery interrupt is implemented by a node, then the set of controls for enabling Error recovery interrupts is IMPLEMENTATION DEFINED. Software uses ERR<n>FR to determine what controls are implemented.

R_{VYFND} For a node <n>, if an Error recovery interrupt is implemented, then a control for enabling the Error recovery interrupt on deferred errors, ERR<n>CTLR.DUI, might be implemented.

R_{XGBJV} For a node <n>, if the ERR<n>CTLR.DUI control is implemented, then the Error recovery interrupt is enabled for deferred errors when ERR<n>CTLR.DUI is `0b1`, and disabled for deferred errors when ERR<n>CTLR.DUI is `0b0`.

R_{KRDFZ} For a node <n>, if the ERR<n>CTLR.DUI control is not implemented, then the Error recovery interrupt is always disabled for deferred errors.

R_{QSYLK} For a node <n>, if an Error recovery interrupt is implemented, then a control for enabling the Error recovery interrupt on uncorrected errors, ERR<n>CTLR.UI, might be implemented.

R_{CBVJB} For a node <n>, if the ERR<n>CTLR.UI control is implemented, then the Error recovery interrupt is enabled for uncorrected errors when ERR<n>CTLR.UI is `0b1` and disabled for uncorrected errors when ERR<n>CTLR.UI is `0b0`.

R_{ZLXWQ} For a node <n>, if the ERR<n>CTLR.UI control is not implemented, then the Error recovery interrupt is always enabled for uncorrected errors.

R_{BLVMZ} For a node <n>, if an Error recovery interrupt is not implemented, then the ERR<n>CTLR.{DUI,UI} controls are not implemented.

I_{HYYWP} For a node <n>, if an error can both signal an in-band error response and be recorded with an component error state of Deferred, and the ERR<n>CTLR.UI control is implemented, then it is recommended that the ERR<n>CTLR.DUI control is also implemented.

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

R_{LMFJX} The Error recovery interrupt is generated when the node records an error, even if the error syndrome is discarded because the Error record already records a higher priority error.

2.5.4 Critical error interrupt

R_{QHJMS} Support for critical error conditions and Critical Error interrupts at a node is IMPLEMENTATION DEFINED. Software uses ERR<n>FR to determine what support is implemented.

R_{LWHDB} Critical Error interrupts provide a mechanism for a node to report a critical error condition to a system controller for error recovery.

I_{WPFSF} An example of a critical error is one where the node has entered a service failure mode which means that the primary error recovery mechanisms cannot be used.

Example 2-6 Example of a critical error

A memory controller enters a failure mode and stops servicing memory requests from application processors, and application processors host the primary error recovery software. The error is signaled to a secondary error controller that has its own private resources in order to log the error.

R_{YQLPR} For a node <n>, if the Critical Error interrupt is implemented, then the Error recovery interrupt is implemented.

R_{LZVMK} For a node <n>, if the Critical Error interrupt is implemented, then the Critical Error interrupt is enabled when ERR<n>CTLR.CI is `0b1` and disabled when ERR<n>CTLR.CI is `0b0`.

R_{JSVFW} For a node <n>, if the Critical Error interrupt is implemented, then when a critical error condition is recorded the node sets ERR<n>STATUS.CI to `0b1`, regardless of whether the Critical Error interrupt is enabled or disabled. ERR<n>STATUS.CI is set to `0b1` in addition to the other syndrome information for the error, which is handled in the normal way.

R_{YMGQG} For a node <n>, if the Critical Error interrupt is implemented and disabled, then when a critical error condition is detected, the node records the component error state as Uncontainable.

I_{BNDZW} Recording the component error state as Uncontainable on a critical error condition when the Critical Error interrupt is disabled has the effect of causing the node to generate an Error recovery interrupt.

I_{VSKSB} For a node <n>, if the Critical Error interrupt is implemented and enabled, then it is IMPLEMENTATION DEFINED how the error is classified at the node.

2.6 In-band error response signaling

R_{QTNMH} For a node <n>, if support for In-band error response signaling, also referred to as External aborts, is implemented by the node, then the control for enabling In-band error response signaling, ERR<n>CTLR.UE, might be implemented. Software uses ERR<n>FR to determine what controls are implemented.

R_{BBFMC} For a node <n>, if the ERR<n>CTLR.UE control is implemented, then In-band error response signaling is enabled when ERR<n>CTLR.UE is `0b1`, and In-band error response signaling is disabled when ERR<n>CTLR.UE is `0b0`.

R_{XDXWP} For a node <n>, if the ERR<n>CTLR.UE control is not implemented and support for In-band error response signaling is implemented, then In-band error response signaling is always enabled.

R_{DMTCY} For a node <n>, if support for In-band error response signaling is not implemented, then the ERR<n>CTLR.UE control is not implemented.

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

R_{JRYXD} When the node signals an In-band error response, it sets ERR<n>STATUS.ER to `0b1`.

2.7 Error record reset

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

R_{LVDPS} The RAS System Architecture defines two classes of reset:

- Cold reset.
- Error Recovery reset.

R_{DKKYC} A Cold reset resets all of the logic in a component, including RAS functionality, to a known initial state.

R_{WXRXD} An Error Recovery reset resets some of the logic in the component to a known state.

However, some state is purposefully unchanged by an Error Recovery reset. Unlike a Cold reset, any recorded error syndrome information is preserved by an Error Recovery reset.

R_{DXBFS} All logic of the component that is reset by an Error Recovery reset is also reset by a Cold reset.

R_{LPJWX} How these resets map to other resets is IMPLEMENTATION DEFINED.

R_{ZLZDR} Mechanisms for asserting resets are IMPLEMENTATION DEFINED.

I_{WFGQQ} R_{ZLZDR} means it is IMPLEMENTATION DEFINED whether it is possible to independently assert an Error Recovery reset and a Cold reset. Arm recommends that an Error Recovery reset can be asserted independently of a Cold reset, and:

- A Cold reset is asserted to a component when it transitions from a powered off state to a powered on state. No state is preserved from the previous powered off state.
- An Error Recovery reset can be asserted at other times, for example when a system fatal error is detected. Error recovery software executed after reset can recover the recorded error syndrome information.

For example, an Error Recovery reset might be implemented by a Warm reset, such as the architectural Warm reset defined for a PE by the Arm architecture. In such an implementation, when Warm reset is asserted, the error records of the component are preserved.

R_{KWZYL} For each message-signaled fault handling, error recovery, and critical error interrupt, the implementation must ensure that assertion of the interrupt is disabled at reset, to prevent a spurious write to a location.

R_{JNSQZ} For each fault handling, error recovery, and critical error interrupt that is not message-signaled, it is IMPLEMENTATION DEFINED whether the assertion of the interrupt is disabled at Cold and/or Error Recovery resets.

I_{VDQFV} Assertion of an interrupt at reset can be disabled by one of the following:

- Implementing the interrupt configuration registers using a recommended layout, and setting the interrupt enable controls, ERR<irq>CR2.IRQEN, to `0b0` at reset. See also R_{GZQWV}.
- Implementing interrupt configuration registers using an IMPLEMENTATION DEFINED layout with an equivalent interrupt enable control that is disabled at reset.
- Defining that each architecturally UNKNOWN reset value for an interrupt enable control in each error record control register ERR<n>CTLR is `0b0`.

Using the interrupt configuration registers in this way is only possible when the error record registers and interrupt configuration registers are reset by the same reset signals.

I_{MBLBG} For many systems, it is recommended that assertion of interrupts is disabled at Cold and Error Recovery resets. However, for some systems this is not necessary or not recommended. For example:

- The interrupts are simple wired interrupts and will be ignored following a reset until software enables the interrupt at the interrupt controller. Software can disable the interrupt at the node before enabling the interrupt at the interrupt controller.
- The system has a requirement that the system must not rely on boot software correctly re-enabling interrupts after a system reset. For example, where a complete system failure must trigger a fail-safe mode of operation.

2.7.1 Error record reset flag

D_{CPRWG} FEAT_RASSA_RV is an OPTIONAL node feature from FEAT_RASSAv1p1.

I_{RNJWF} FEAT_RASSA_RV allow software to determine, during boot, whether an error recorded by the error record occurred before or after an Error Recovery reset.

R_{SMQPN} When FEAT_RASSA_RV is implemented by a node, each Error record <n> owned by the node includes the *reset valid* flags, ERR<n>STATUS.{RV, RV2}.

S_{SKYFL} FEAT_RASSA_RV is identified to software for node <n> by ERR<n>FR.RV.

R_{GGTCT} ERR<n>STATUS.{RV, RV2} are R/W1C fields.

2.7.2 Reset values

I_{PQVFQ} When the node records an error in an Error record, depending on the type of error being recorded, it is IMPLEMENTATION DEFINED whether some fields are set to a zero or unchanged.

In most cases, for each of these fields, it is IMPLEMENTATION DEFINED which of the following applies:

- The node sets the field to zero on Cold reset, meaning the value is not required to be changed when the first error is recorded.
- The node sets the field to zero on recording the first error after Cold reset.

To allow for either implementation, software must clear these fields to zero after logging a recorded error and performing a software reset of the Error record.

For more information, see *Accessing ERR<n>STATUS* in ERR<n>STATUS.

I_{ZTWHG} The reset values of ERR<n>STATUS.{AV,V,MV}, ERR<FirstRecordOfNode(n)>CTLR.ED, and ERR<n>PFGCTL.CDNEN depend on whether FEAT_RASSA_SRV is implemented.

D_{MCHNV} FEAT_RASSA_SRV is an OPTIONAL node feature from FEAT_RASSAv1p1.

S_{GHQHP} FEAT_RASSA_SRV is identified to software for node <n> by ERR<n>FR.SRV.

I_{MKKGG} When FEAT_RASSA_SRV is not implemented by a node <n>, all of the following apply:

- ERR<n>STATUS.{AV, V, MV} are set to {0b0, 0b0, 0b0} on a Cold reset and preserved on an Error Recovery reset.
- ERR<FirstRecordOfNode(n)>CTLR.ED is set to an IMPLEMENTATION DEFINED value, 0b0 or 0b1, on a Cold reset and preserved on an Error Recovery reset.
- If the Common Fault Injection Model Extension is implemented, ERR<n>PFGCTL.CDNEN is set to 0b0 on a Cold reset and preserved by an Error Recovery reset.
- When FEAT_RASSA_RV is implemented, ERR<n>STATUS.{RV, RV2} are set to {0b1, 0b1} on both Cold reset and Error Recovery reset.

This means that:

- If FEAT_RASSA_RV is implemented, there are two reset signals for the node.
- Otherwise, there is effectively a single reset signal for the node.

R_{LCDPD} When FEAT_RASSA_SRV is implemented by a node <n>, all of the following apply:

- ERR<n>STATUS.{AV, V, MV} are set to architecturally UNKNOWN values on a Cold reset and preserved on an Error Recovery reset.
- ERR<FirstRecordOfNode(n)>CTLR.ED is set to 0b0 on both Cold resets and Error Recovery resets.
- If the Common Fault Injection Model Extension is implemented, ERR<n>PFGCTL.CDNEN is set to 0b0 on both Cold resets and Error Recovery resets.
- When FEAT_RASSA_RV is implemented, ERR<n>STATUS.{RV, RV2} are set to {0b1, 0b1} on both Cold reset and Error Recovery reset.

This means there is effectively a single reset signal for the node.

S_{DVBVS} If ERR<n>STATUS.V is reset to an architecturally UNKNOWN value then power-on reset software must initialize ERR<n>STATUS before enabling error detection and recording by the node. If error detection and recording is enabled when ERR<n>STATUS.V is still architecturally UNKNOWN, then any detected error might be discarded because of the error record overwriting rules described in the sections Prioritizing errors, FEAT_RASSAv1 and Prioritizing errors, FEAT_RASSAv1p1.

2.8 Extensions

2.8.1 The RAS Timestamp Extension

R_{BWYMJ} The RAS Timestamp Extension is an optional part of FEAT_RASSAv1p1.

I_{PZVXP} The RAS Timestamp Extension provides a standard mechanism for timestamping Error records.

R_{TRHJP} For a given Error record <n>, if the RAS Timestamp Extension is implemented, the timestamp value is recorded in ERR<n>MISC3.

S_{FZJMJ} Software uses ERR<n>FR.TS to determine whether the RAS Timestamp Extension is implemented by node <n>.

R_{MHTSQ} The timestamp value uses either the system Generic Timer counter or an IMPLEMENTATION DEFINED timebase.

S_{HLDRQ} Software uses ERR<n>FR.TS to determine which timebase is used by node <n>.

R_{XKBJS} Other than when IMPLEMENTATION DEFINED conditions apply, the following are true:

- The timebase is encoded as a plain binary number.
- The timebase is monotonically increasing at a fixed rate compared to wallclock time.

I_{PQCNK} The IMPLEMENTATION DEFINED conditions are to allow for the timebase to violate these conditions during initial system configuration.

2.8.2 The Common Fault Injection Model Extension

R_{CVLDN} The Common Fault Injection Model Extension is an optional part of FEAT_RASSAv1p1.

I_{TSWKX} Other forms of error or fault injection are permitted. For example, if the Common Fault Injection Model Extension is not implemented, the ERRIMPDEF<m> registers might be used for an IMPLEMENTATION DEFINED fault injection mechanism.

R_{YBSBX} The Common Fault Injection Model Extension can only be implemented for Error records accessed through a memory-mapped group of Error records if ERRDEVARCH.REVISION >= `0b0001`.

I_{PTGZW} The Common Fault Injection Model Extension fakes the detection of an error at a component.

I_{CPYFQ} A faked error detection results in the node signaling the appropriate ones of the Fault handling interrupt, Error recovery interrupt, and In-band error response, according to the type of injected error and the control settings of the node.

R_{XGXNW} If FEAT_RME is implemented:

- When all the following are true, it must not be possible to determine which address a PE is accessing from an address captured in ERR<n>ADDR as a result of error injection:
  - The ERR<n>ADDR is accessible to PEs not in Root state.
  - The error injection controls are accessible to PEs not in Root state.

If FEAT_RME is not implemented, Arm recommends:

- When all the following are true, it must not be possible to determine which address a PE is accessing from an address captured in ERR<n>ADDR as a result of error injection:
  - The ERR<n>ADDR is accessible to PEs not in Secure state.
  - The error injection controls are accessible to PEs not in Secure state.

I_{FRCGL} To achieve R_{XGXNW}, it is permitted for an implementation to, for example:

- Not provide a valid address.
- Not update the ERR<n>ADDR as a result of error injection.

R_{RYFQP} The Common Fault Injection Model Extension supports generating a subset of the component error state types supported by the node.

IYSQHB Arm recommends that the Common Fault Injection Model Extension supports all the component error state types supported by the node.

SQVLPN The Common Fault Injection Model Extension is identified to software for node <n> by ERR<n>FR.INJ,

SZBZHW Software uses ERR<n>PFGF to determine the Common Fault Injection Model Extension capabilities for node <n> that implements the Common Fault Injection Model Extension.

IZDPWF If a node is not capable of recording a component error state type, then it does not support injecting that component error state type.

IYMWNF The Common Fault Injection Model Extension registers are:

- ERR<n>PFGF.
- ERR<n>PFGCTL.
- ERR<n>PFGCDN.

The Common Fault Injection Model Extension registers are not accessible from AArch32 state. However, when accessed via ERXFR, AArch32 state can access the ERR<n>FR.INJ field described in this section.

IJFHGG The Common Fault Injection Model Extension registers can be implemented directly, or as a Fault Injection Group. See Fault injection groups.

IQFYWD Additional constraints might apply if fault injection can affect the operation of Secure and/or Root states.

ITQVQW See also Security and Virtualization.

2.8.2.1 Operation of the Common Fault Injection Model Extension

DYHZHK The behaviors in this section apply for a given node <n> if node <n> implements the Common Fault Injection Model Extension.

RVDZSG When software writes `0b1` to ERR<n>PFGCTL.CDNEN:

- The internal Error Generation Counter is set to ERR<n>PFGCDN.CDN if all of the following apply:
  - Error reporting and logging at the node is enabled.
  - ERR<n>PFGCTL.CDNEN was previously `0b0`.
  - ERR<n>PFGCDN.CDN is nonzero.
  - The component is not in the fault injection state.
- Otherwise, all of the following apply:
  - It is UNPREDICTABLE whether the Error Generation Counter is unchanged or is set to ERR<n>PFGCDN.CDN, which might be zero.
  - If the component is in the fault injection state, the component might leave the fault injection state.
  - If the component is not in the fault injection state, the component might enter the fault injection state.

IXDWGY The current value of the Error Generation Counter is not visible to software.

RPLZMT If error reporting and logging at the node is enabled, then while ERR<n>PFGCTL.CDNEN is `0b1` and the Error Generation Counter is nonzero, the Error Generation Counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate.

RPQTKT If error reporting and logging at the node is disabled, then while ERR<n>PFGCTL.CDNEN is `0b1` and the Error Generation Counter is nonzero, it is IMPLEMENTATION DEFINED whether:

- The Error Generation Counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate.
- The Error Generation Counter remains unchanged.

IDMNZX The rate at which the component decrements the counter is defined by the component. For example, it might be the native clock rate for the component, and this might not be the same as the PE clock rate. Software typically discovers this rate from firmware.

RDDPMH When the Error Generation Counter decrements to or past zero, the component enters a fault injection state.

R_{YXXWT} When the component is in the fault injection state, the component does all of the following:

- Fakes detection of the component error state type(s) described by ERR<n>PFGCTL.
- Reports the injected error to the node.
- If error reporting and logging at the node is enabled in ERR<n>CTLR.ED, then the node recorded the injected error.
- If error reporting and logging at the node is disabled in ERR<n>CTLR.ED, then it is UNPREDICTABLE whether or not the node records the injected error.

It is IMPLEMENTATION DEFINED whether this occurs only on the next access to the component in the fault injection state, or occurs spontaneously in the fault injection state. ERR<n>PFGF.NA describes which.

The component then leaves fault injection state.

I_{DNMXX} For components that support the concept of an *access to the component*, Arm recommends that R_{YXXWT} applies on the next access to the component.

R_{YFBLN} If ERR<n>PFGCTL.CDNEN is cleared to `0b0` when the component is in the fault injection state, it is UNPREDICTABLE whether the component leaves the fault injection state or remains in the fault injection state.

R_{XMZBB} When an injected error is recorded, the node signals the appropriate ones of the Fault handling interrupt, Error recovery interrupt, and In-band error response, according to the type of injected error and the control settings of the node.

R_{GJXGL} When an injected error is recorded, the node writes the ERR<n>STATUS.{V, UE, CE, DE, UET} fields according to the component error state type described by ERR<n>PFGCTL.

R_{TSXMT} If ERR<n>PFGCTL defines multiple component error state types, or none, then the behavior is UNPREDICTABLE and is one of the following:

- No error is injected.
- An error is injected with an UNPREDICTABLE choice of component error state.

R_{XLJMM} It is IMPLEMENTATION DEFINED how the node updates the ERR<n>STATUS.{AV, ER, OF, MV, PN, CI, IERR, SERR}, ERR<n>ADDR, and ERR<n>MISC<m> when recording an injected error. ERR<n>PFGF describes the IMPLEMENTATION DEFINED options and the controls available in ERR<n>PFGCTL.

I_{CSSDM} For many fields, the implementation has the choice to either set the syndrome register or field according to the access that triggers the injected error, or provide finer-grained control over the field, either by a control bit in ERR<n>PFGCTL or by not updating the register or field when the injected error is recorded meaning software can write the injected syndrome to the register or field ahead of injecting the error.

R_{WMDWR} For each of the ERR<n>STATUS.{CI, ER, PN} bits, the behavior is UNPREDICTABLE if all of the following are true:

- ERR<n>PFGF defines that the value injected is controlled by the corresponding ERR<n>PFGCTL bit.
- The corresponding ERR<n>PFGCTL bit is `0b1`.
- For the ER and PN bits, the definition of the ERR<n>STATUS field defines that the bit is not valid for the component error state requested by ERR<n>PFGCTL. For the CI bit, the component error state requested by ERR<n>PFGCTL is not one of an IMPLEMENTATION DEFINED set of permitted values for critical error conditions.

The UNPREDICTABLE behavior is one of:

- No error is injected.
- An error is injected, but the component error state and syndrome bits do not match the requested error type.
- The error is injected as requested, including setting the invalid bit or bits to the requested values.

I_{QSLVZ} This means that:

- It is IMPLEMENTATION DEFINED which component error states the CI value can be injected with.
- The PN value can be injected with a component error state of Uncorrected or Deferred and cannot be injected with a component error state of Corrected.
- The ER value can be injected with a component error state of Uncorrected and cannot be injected with a component error state of Corrected.
- It is IMPLEMENTATION DEFINED whether the ER value can be injected with a component error state of Deferred.

R_{GGFSF} If a single node has multiple Error records, then only the first Error record has fault injection registers.

R_{RBYRG} If a single node has multiple Error records and any of ERR<n>PFGF.{SYN, AV, MV} for the first Error record of the node are nonzero, meaning the fault injection mechanism does not update all or some of the ERR<n>MISC<m> or fields when the injected error is recorded, then the injected fault is recorded in the first Error record. Otherwise, the injected error might be recorded in any of the multiple Error records.

Note

If a single node has multiple Error records and any of ERR<n>PFGF.{SYN, AV, MV} for the first Error record of the node are zero then a node might define which Error record is updated or implement an IMPLEMENTATION DEFINED control to allow this to be specified.

I_{BDDZZ} If the node implements Fault handling interrupt, Error recovery interrupt, and Critical Error interrupt as edge-triggered interrupts, then recording an injected error has the same behavior as recording a detected error, for generating the edge-triggered interrupt. That is, the interrupt is generated if the interrupt is enabled for the type of error being injected.

I_{TVRDH} If the node implements Fault handling interrupt, Error recovery interrupt, and Critical Error interrupt as level-sensitive interrupts, then the level of the interrupt request is a function of the values of the control and status register fields. The behavior of the interrupt request does not depend on whether the control and status registers were written by the node when detecting an error, or written by error injection.

R_{CFTGZ} If the Error Generation Counter is zero and ERR<n>PFGCTL.R is 1, then:

- If ERR<n>PFGCDN.CDN is nonzero, then the internal Error Generation Counter is set to ERR<n>PFGCDN.CDN.
- If ERR<n>PFGCDN.CDN is zero, the behavior is UNPREDICTABLE and is one of:
  - The Error Generation Counter is unchanged.
  - The Error Generation Counter is set to zero.
  - The Error Generation Counter is set to zero and the component reenters the fault injection state.

2.8.3 Standard format Corrected error counter

I_{FQLPT} The RAS System Architecture defines standard formats for a Corrected error counter. Software uses ERR<n>FR to determine whether any standard format Corrected error counter is implemented by a node.

R_{XYFVB} If a standard format Corrected error counter is implemented by a node, then it is IMPLEMENTATION DEFINED whether a single counter or a pair of counters is implemented by Error records owned by the node.

R_{SLPQW} For an Error record <n>, if a standard format Corrected error counter is implemented by the node and the error record can record countable errors, then the counter or counters are recorded in ERR<n>MISC0.

R_{BYDBW} It is IMPLEMENTATION DEFINED whether an Error record can record countable errors.

I_{FYBWQ} If a pair of standard format Corrected error counters are implemented by a node, then the node provides all of the following:

- A first (repeat) error counter to count the first error and any subsequent error detected at the same location.
- A second (other) error counter to count errors detected in other locations.

R_{GYPDJ} If a pair of standard format Corrected error counters are implemented by a node, then an Error record <n> records a counted-fault location for the error, in one or more of:

- The ERR<n>ADDR register.
- The ERR<n>STATUS.IERR field.
- The ERR<n>STATUS.SERR field.
- The ERR<n>MISC<m> registers.

It is IMPLEMENTATION DEFINED which of these or parts thereof describe the counted-fault location.

Note: These registers might contain additional IMPLEMENTATION DEFINED fault location information that is not considered part of the counted-fault location.

R_{JMTCG} The counted-fault location recorded in Error record <n> is either valid or invalid.

R_{JCNNX} If the counted-fault location or part of the counted-fault location is held in the ERR<n>ADDR register, then all of the following apply:

- This part is valid when ERR<n>STATUS.{V, AV} is {0b1, 0b1}.
- It is IMPLEMENTATION DEFINED whether this part of the counted-fault location is treated as valid or invalid when ERR<n>STATUS.{V, AV} is {0b1, 0b0}.
- This part is invalid when ERR<n>STATUS.V is 0b0.

R_{JMVKQ} If the counted-fault location or part of the counted-fault location is held in the ERR<n>STATUS.IERR field, then this part is valid when ERR<n>STATUS.V is 0b1 and invalid otherwise.

R_{LTFXM} If the counted-fault location or part of the counted-fault location is held in the ERR<n>STATUS.SERR field, then this part is valid when ERR<n>STATUS.V is 0b1 and invalid otherwise.

R_{SLYKF} If the counted-fault location or part of the counted-fault location is held in the ERR<n>MISC<m> registers, then:

- This part is valid when ERR<n>STATUS.{V, MV} is {0b1, 0b1} and IMPLEMENTATION DEFINED parts of the syndrome data indicate the registers contain a valid counted-fault location.
- It is IMPLEMENTATION DEFINED whether this part of the counted-fault location is treated as valid or invalid when ERR<n>STATUS.{V, MV} is {0b1, 0b0}.
- This part is invalid when ERR<n>STATUS.V is 0b0.

R_{LSTYJ} If the counted-fault location is held across multiple of these registers, then the counted-fault location is valid only if all parts are valid and invalid otherwise.

I_{QDPMP} The counted-fault location is always invalid if ERR<n>STATUS.V is 0b0, that is, if no error has been recorded by the Error record since ERR<n>STATUS.V was last cleared to 0b0.

I_{GGWGP} The content of IMPLEMENTATION DEFINED syndrome is IMPLEMENTATION DEFINED. This permits, but does not require, for example, the ERR<n>MISC<m> registers to contain additional valid flags for other parts of the syndrome, or for some parts of ERR<n>MISC<m> to be valid only for some values of ERR<n>STATUS.{IERR, SERR}.

I_{WRGPQ} For some implementations, ERR<n>ADDR is always written when an error is recorded, meaning the hardware never sets ERR<n>STATUS.{V, AV} to {0b1, 0b0} when recording an error. Similarly, for some implementations, the hardware never sets ERR<n>STATUS.{V, MV} to {0b1, 0b0} when recording an error. For these cases, the implementation might ignore the applicable one or ones of the AV and MV bits when determining whether the counted-fault location is valid.

R_{JQZZT} If a pair of standard format Corrected error counters are implemented by a node, then when a countable error is recorded by Error record <n>:

- The first (repeat) error counter counts an error if either of the following are true:
  - The counted-fault location recorded in error record <n> is invalid.
  - The error being counted is at the same location as the valid counted-fault location recorded in error record <n>.
- The second (other) counter counts the error otherwise.

I_{BYGGW} When the counted-fault location recorded in error record <n> is invalid, because this typically means that ERR<n>STATUS.V is 0b0, the node typically overwrites the syndrome, meaning it captures the new counted-fault location. Otherwise, because ERR<n>STATUS.V is 0b1 the node keeps the syndrome, meaning the counted-fault location is unchanged.

R_{FYCFY} If a standard format Corrected error counter is implemented by a node, then if counting an error causes unsigned overflow of the Corrected error counter:

- The counter overflow flag is set to 0b1.
- A Corrected error event occurs.

I_{QZJFY} IMPLEMENTATION DEFINED forms of counters, including other sizes, other overflow models, and other miscellaneous syndrome register locations, might be implemented.

I_{YTTKW}

See also:

- Writing the error record.
- Fault handling interrupt.

2.9 Accessing RAS registers

I_{MQDMJ} RAS registers summary defines the registers for memory-mapped Error records.

I_{PVYZG} *Error record System register view* in *Arm® Architecture Reference Manual, for A-profile architecture* defines System registers for accessing a group of Error records.

R_{HQQNS} It is IMPLEMENTATION DEFINED which components in the system, if any, implement memory-mapped Error records.

R_{WWDBV} A memory-mapped component might implement several Error records in an Error record group, relating to one or more nodes.

D_{XPBLL} The RAS System Architecture defines the reusable formats described in this section for memory-mapped Error records.

R_{DHYDC} The Table 3-4 describes a format for a memory-mapped component that implements a single Error record. This might be implemented as part of the control registers for a memory-mapped component. In this format, the first register, ERR0FR, is at an address aligned to a multiple of 64 bytes.

The Table 3-4 might be repeated in the control registers for a memory-mapped component that implements a small number of Error records. Each error record has its own IMPLEMENTATION DEFINED base within the control registers of the component.

R_{PCXRD} The Error records in a memory-mapped component might be accessible only through that component, or might be shared and accessible through any of:

- System registers by one or more PEs.
- Other memory-mapped components in the same physical address space, including aliases with the same Error record group.
- Other memory-mapped components in other address spaces. For example, in both Non-secure and Secure physical address spaces.

I_{JFZRW} Arm recommends that each memory-mapped Error record is accessible at most once in any given physical address space.

2.9.1 Error record groups

D_{QJRWD} When FEAT_RASSAv2 is implemented, each Error record group and, if applicable, its corresponding fault injection group, implements one of:

- FEAT_RASSA_4KB_GRP, meaning it supports the 4KB Error record group format. FEAT_RASSA_4KB_GRP is permitted from FEAT_RASSAv1.
- FEAT_RASSA_16KB_GRP, meaning it supports the 16KB Error record group format. FEAT_RASSA_16KB_GRP is permitted from FEAT_RASSAv2.
- FEAT_RASSA_64KB_GRP, meaning it supports the 64KB Error record group format FEAT_RASSA_64KB_GRP is permitted from FEAT_RASSAv2.

When FEAT_RASSAv2 is not implemented, an Error record group implements FEAT_RASSA_4KB_GRP.

R_{TPFWF} The first register in an Error record group, ERR0FR, is aligned to a multiple of:

- 4KB, when FEAT_RASSA_4KB_GRP is implemented.
- 16KB, when FEAT_RASSA_16KB_GRP is implemented.
- 64KB, when FEAT_RASSA_64KB_GRP is implemented.

S_{NVRSW} When FEAT_RASSAv2 is implemented, for each Error record group and each fault injection group, implementation of one of FEAT_RASSA_4KB_GRP, FEAT_RASSA_16KB_GRP, or FEAT_RASSA_64KB_GRP is identified to software by ERRPIDR4.SIZE.

If ERRPIDR4 is not implemented, software identifies the group format by IMPLEMENTATION DEFINED means.

S_{NBFYF} In an Error record group, the group is described to software by the following registers:

- The following registers provide a unique combination of a part number identifier, revision, and designer of the group:

— The implementation identification register, ERRIIDR. This register is optional.
— The peripheral identification registers ERRPIDR0, ERRPIDR1, ERRPIDR2, ERRPIDR3, and ERRPIDR4. These registers are optional.

Arm recommends that at least one of these identification mechanisms is implemented.

- The ERRDEVARCH register defines that the group implements the RAS System Architecture and the version implemented.
- The optional ERRDEVAFF register describes when the group records errors for components that have an *affinity* with a single PE, or a group of PEs in the system.
  Each PE has a unique value that identifies it in the system. MPIDR_EL1 in the PE and ERRDEVAFF in the Error record group contain this value. ERRDEVAFF might contain a value that matches a group of PEs. Arm deprecates use of ERRDEVAFF by software.
- ERRDEVID identifies the highest numbered index of the Error records that can be accessed, and what features are implemented by the Error record group.

I_{GFLXS} The maximum number of Error records that can be accessed depends on which of FEAT_RASSA_4KB_GRP, FEAT_RASSA_16KB_GRP, or FEAT_RASSA_64KB_GRP is implemented. If the Common Fault Injection Model Extension is implemented and fault injection groups are not implemented, fewer Error records will be accessible.

Common Fault Injection Model Extension implemented and Fault Injection Groups not implemented	Feature Implemented	Number of accessible Error records
No	FEAT_RASSA_4KB_GRP	≤56
No	FEAT_RASSA_16KB_GRP	≤224
No	FEAT_RASSA_64KB_GRP	≤896
Yes	FEAT_RASSA_4KB_GRP	≤24
Yes	FEAT_RASSA_16KB_GRP	≤96
Yes	FEAT_RASSA_64KB_GRP	≤384

R_{YGWDK} In an Error record group, each Error record occupies a set of locations at offsets from an Error record base. This Error record base is offset from the group base at a multiple of 64 times the index of the Error record.

R_{YFCNK} When FEAT_RASSA_4KB_GRP is implemented, the Error record group includes a group status register, ERRGSR.
When FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, the Error record group includes multiple group status registers, ERRGSR<m>, where ERRGSR<m> defines the group status for error records [(64×m)..(64×m+63)].

R_{CKXQC} When FEAT_RASSAv2 is not implemented, it is IMPLEMENTATION DEFINED whether the status of each error record <n> supports being read through the group status register.
When FEAT_RASSAv2 is implemented, a group status register bit ERRGSR<n DIV 64>[n MOD 64] is implemented for each error record <n> in the Error record group.

I_{WLXTV} A read from ERRGSR<m> requires the component to collect all ERR<n>STATUS.V values for Error records [(64×m)..(64×m+63)] into a single return value. This allows software to check the status of up to 64 error records in a single read.
When the component aggregates error records for multiple other components into a single view, this might involve many accesses into those other components. When multiple ERRGSR<m> registers are implemented, because a read of ERRGSR<m> only accesses ERR<n>STATUS.V values for error records [(64×m)..(64×m+63)], one implementation choice might be to assign error record indices so as to reduce the number of remote accesses that need to be made. However, the advantage to software of using ERRGSR<m> scales with the number of status bits returned with each read. For example, if a single record is mapped to the ERRGSR<m> register, then there is no advantage for software over reading the ERR<n>STATUS register directly.

2.9.2 Fault injection groups

I_{TJYGF} The Common Fault Injection Model Extension is not supported in the Table 3-4 format.

D_{FQHBV} FEAT_RASSA_PFG_GRP is an OPTIONAL Error record group feature from FEAT_RASSAv1p1.

I_{CCQHP} FEAT_RASSA_PFG_GRP allows for fault injection controls to be separated from error records, so that software with access to the latter does not necessarily have access to the former.

R_{RGWJK} When FEAT_RASSA_PFG_GRP is implemented by an Error record group, the Common Fault Injection Model Extension registers for any node that is part of an Error record group implementing the Common Fault Injection Model Extension are accessed through a fault injection group.

S_{XMPMK} FEAT_RASSA_PFG_GRP is identified to software by ERRDEVID.PFG.

D_{XCNYR} A fault injection group is a component comprising the Common Fault Injection Model Extension registers for a corresponding Error record group.

S_{FSZQF} A fault injection group is identified to software by ERRDEVARCH.

S_{JVCXF} If the ERRDEVAFF register is implemented, then it has the same value for both the Error record group and the fault injection group.

I_{FKNGJ} If FEAT_RASSA_PFG_GRP is not implemented, the Common Fault Injection Model Extension registers might be implemented directly. See The Common Fault Injection Model Extension.

2.9.3 System RAS Agents

D_{YDPRQ} A RAS agent implements an Error record group and provides RAS error control and reporting for other components.

RAS agents can be hierarchical. That is, a first RAS agent reports errors to a second RAS agent. In this case, the first RAS agent (called the downstream RAS agent) records the error, but signals the second RAS agent (called the upstream RAS agent) that it has a valid RAS record. This then repeats to create a hierarchy.

The RAS agent at the top of such a tree is called a System RAS agent. The System RAS agent is connected to the interrupt controller.

I_{ZMCXR} Figure 2-2 shows an example system:

- Containing multiple components, IP_n.
- Where each IP_n contains *error detection* logic, ED_n.
- Where the error detection logic reports errors to the RAS agent, RA_n.

The RAS agents report the error hierarchically to the System RAS agent, RA_0, using the interfaces between RAS agents, shown as **FAULT**.

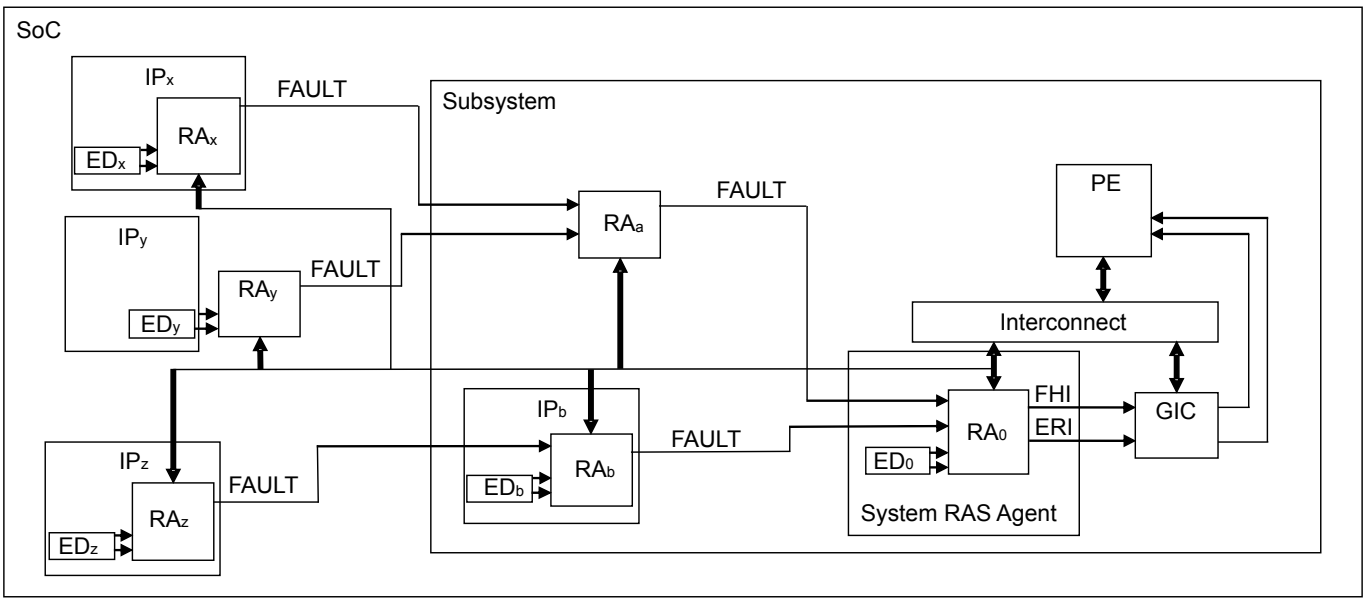


Figure 2-2 Example RAS Agents

Figure 2-3 shows an example system where **FAULT** is comprised of the following signals:

- **VALID** is asserted when the downstream RAS agent contains at least one valid error record. That is:
  - If the downstream RAS agent implements a single error record, **VALID** is asserted when ERR0STATUS.V is 1.
  - If the downstream RAS agent implements an Error record group, **VALID** is asserted when ERRGSR is not zero. Figure 2-4 shows this.
- **FHI** is the Fault handling condition status from the downstream RAS agent.
- **ERI** is the Error recovery condition status from the downstream RAS agent.
- **CRI** is the Critical Error condition status from the downstream RAS agent.

FHI, **ERI**, and **CRI** are optional signals. A signal is not implemented if the downstream RAS agent does not implement its corresponding condition. For example, in Figure 2-3 the Critical Error condition is not implemented.

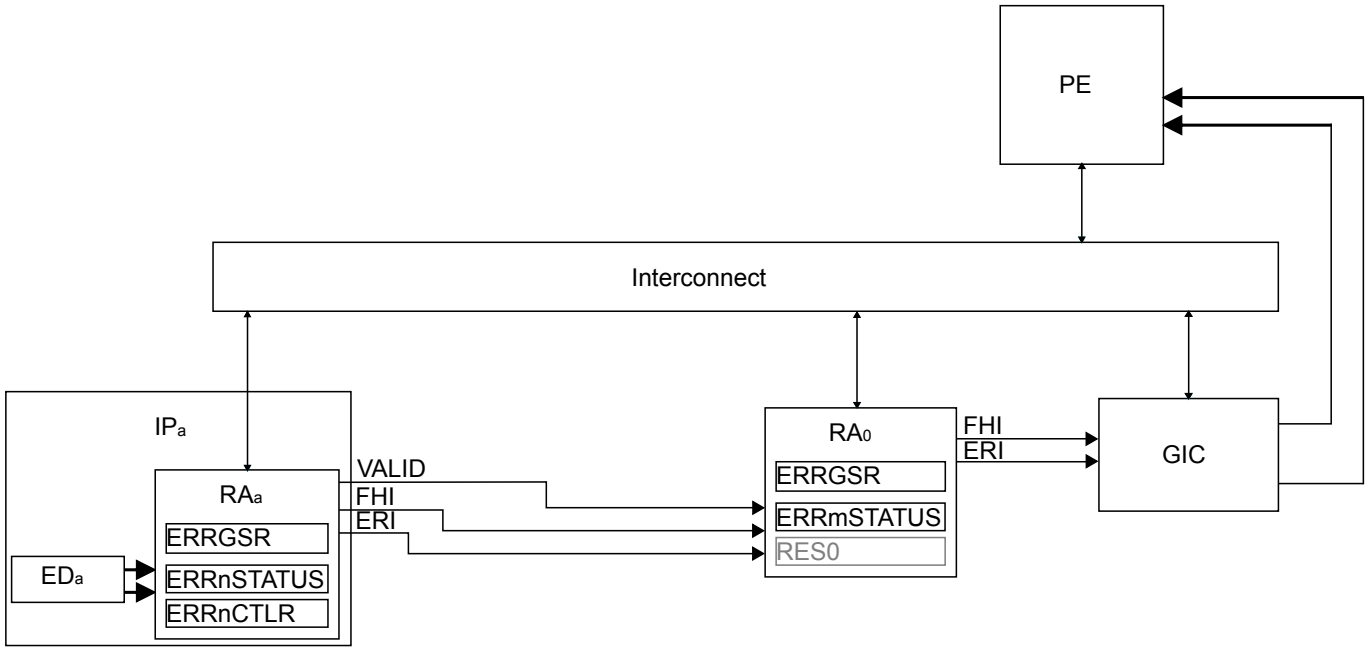


Figure 2-3 FAULT connection between RAS agents

Figure 2-3 shows that RA_0 implements a proxy error record for RA_a, ERRmSTATUS. RA_0 has no other error records in Figure 2-3, however RA_0 is permitted to contain a mix of proxy error records and error records for other components, as shown in Figure 2-2.

RA_0 is connected to the GIC for generating interrupts. The connection to a GIC is either directly wired, or by generating MSIs. There might be connections to other interrupt controllers for out-of-band reporting.

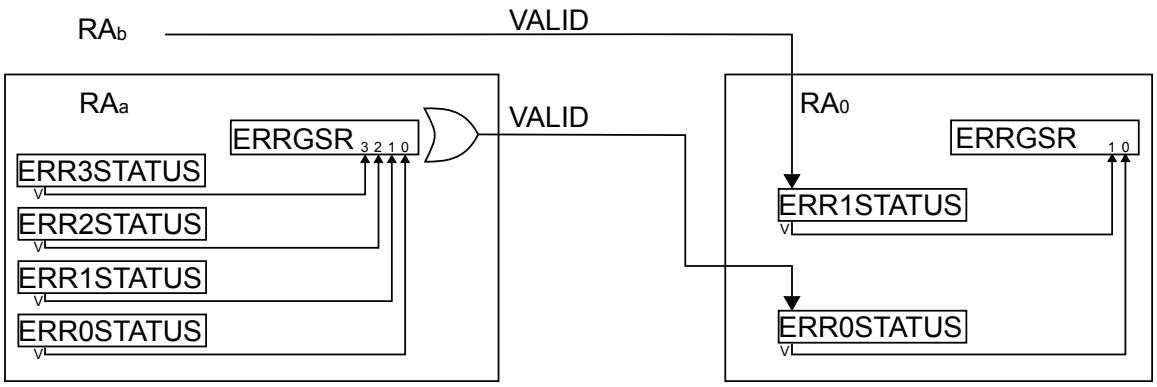


Figure 2-4 VALID signal between RAS agents

Figure 2-4 shows that the **VALID** signal for RA_a is derived from the value of ERRGSR in RA_a, and that the proxy error record 0 in RA_0 therefore shows the status for the whole of RA_a. In this figure, RA_a has four error records, 0 to 3.

A second **VALID** signal for RA_b is similarly derived from the value of ERRGSR in RA_b, and proxy error record 1 in RA_0 shows the status for the whole of RA_b. The details of RA_b are not shown, but it might also have multiple error records.

Therefore in RA_0, ERRGSR[0] shows the status for the whole of RA_a and ERRGSR[1] shows the status for the whole of RA_b.

Only these two error records are shown for RA_0, although in practice RA_0 would contain multiple proxy error records, and can contain a mix of proxy and other error records.

Figure 2-4 does not show the interrupt signals.

I_{LYRFR} RAS agents reduce the cost of finding an active error record to a log(n) search. Each layer indicates the status of up-to 64 lower levels through a single 64-bit ERRGSR<m> status register.

I_{FJVMZ} Per-node error detection and interrupt controls (ERR<n>CTLR.{DFI, CFI, FI, DUI, UI}) and error records are implemented by the downstream RAS agent connected to the component that detects the error.

The downstream RAS agent might implement FEAT_RASSA_IRQCR_SIMPLE to control signaling error statuses on the interface to the upstream RAS agent. Any interrupt configuration registers that control signaling interrupts to the interrupt controller are implemented by the System RAS agent.

This allows software the same fine-grain control of which nodes generate interrupts, and provides an additional level of control and merging of interrupt sources at the upstream RAS agent.

I_{HGJRX} The downstream RAS agent might include Common Fault Injection Model Extension registers, as permitted by FEAT_RASv1p1, or might implement a complementary fault injection group.

I_{JTSVG} The downstream RAS agent might include ERRIMPDEF<m> IMPLEMENTATION DEFINED registers, as permitted by FEAT_RASv1p1.

R_{BZMHP} A downstream RAS agent appears in the programmers' model of the upstream RAS agent as occupying a proxy error record. For the proxy error record <n> in the upstream RAS agent:

- ERR<n>FR.{ED, ERT} indicates that the record is a proxy for a downstream RAS agent.
- ERR<n>STATUS.{V, ERI, FHI, CRI} are read-only flags which report the downstream RAS agent error record error status:
  - ERR<n>STATUS.V reads as 1 if, for any error record <p> in the downstream RAS agent, ERR<p>STATUS.V is 1, and ERR<n>STATUS.V reads as 0 otherwise.
  - ERR<n>STATUS.ERI reads as 1 if the Error recovery condition is true for any error record <p> in the downstream RAS agent, and ERR<n>STATUS.ERI reads as 0 otherwise.
  - ERR<n>STATUS.FHI reads as 1 if the Fault handling condition is true for any error record <p> in the downstream RAS agent, and ERR<n>STATUS.FHI reads as 0 otherwise.
  - ERR<n>STATUS.CRI reads as 1 if the Critical Error condition is true for any error record <p> in the downstream RAS agent, and ERR<n>STATUS.CRI reads as 0 otherwise.

  Reset of the proxy error record does not affect ERR<n>STATUS.{V, ERI, FHI, CRI}.
- ERR<n>CTLR, ERR<n>ADDR, ERR<n>MISC<m>, and the remaining fields in ERR<n>STATUS are RES0.

For the upstream RAS agent, ERRGSR<n DIV 64>[n MOD 64] is an alias for ERR<n>STATUS.V.

R_{RBKZK} The upstream RAS agent implements FEAT_RASSA_PARENT_GRP.

A RAS agent that does not implement FEAT_RASSA_PARENT_GRP does not include proxy error records.

D_{TJQKB} When FEAT_RASSA_PARENT_GRP is implemented by a RAS agent, FEAT_RASSA_ERT is implemented by each node <n> in the RAS agent, and, if ERR<n>FR.ED is `0b11`, then ERR<n>FR.ERT defines the *error record type*.

I_{FNPWG} Because the downstream RAS agent and upstream RAS agent are separate peripherals in the system, the architecture does not guarantee the order in which accesses are completed. This includes two accesses from the same PE to Device-nGnRnE memory, and the indirect writes of a reset.

In addition, the link between the downstream and upstream RAS agents is a separate path from the paths used by a PE to access the RAS agent registers.

For more information, see *Ordering relations* in *Arm® Architecture Reference Manual, for A-profile architecture*.

Example 2-7 Example showing uncertain ordering of downstream RAS agent and upstream RAS agent accesses

Considering the following sequence:

1. PE A performs a sequence of writes W_1, consisting of a write to each ERR<n>STATUS register in a downstream RAS agent B that clears ERR<n>STATUS.V to `0b0`.
2. PE A executes an instruction or sequence of instructions R_2 to ensure that W_1 is complete. For example, one of the following:
   - R_2 is a sequence of reads of ERR<n>STATUS in B, each in program order after the corresponding write in W_1.
   - R_2 is a read of ERRGSR in B, that is ordered after W_1 and observes that ERRGSR[n] is `0b0`.
3. PE A performs a read R_3 of ERR<m>STATUS in the proxy error record <m> for B in an upstream RAS agent C, such that R_2 is Ordered-before R_3.

Even if no other fault is recorded by B after W_1, I_{FNPWG} means that R_3 might not return ERR<m>STATUS.V in C equal to `0b0` as expected. This is similar to the issue of deasserting an interrupt request at a peripheral and that interrupt still being observed as asserted at an interrupt controller.

S_{CFTWB} The topology that maps error record <n> in the upstream RAS agent to a downstream RAS agent is described to software through firmware tables. There is no description of this mapping nor hardware support for topology detection in the architecture.

2.9.4 Access requirements for memory-mapped views of RAS error records

D_{DRNNK} The requirements for a memory-mapped view of RAS Error records are described in *Requirements for Memory-mapped Components* in *Arm® Architecture Reference Manual, for A-profile architecture*.

Chapter 3
RAS Memory-mapped Register Descriptions

This chapter describes the RAS memory-mapped registers. It contains the following sections:

- RAS registers summary.
- RAS register descriptions.

3.1 RAS registers summary

This section describes the memory-mapped interface to the RAS registers. The descriptions in this section apply whether the error records is accessed:

- Through the indirection mechanism, as described in *Error record System register view* in *Arm® Architecture Reference Manual, for A-profile architecture*.
- As memory-mapped registers, as described in RAS memory-mapped register views.

3.1.1 RAS memory-mapped register views

The following tables show the memory-mapped view of the RAS registers:

- Table 3-1 for the error record group memory-mapped registers when FEAT_RASSA_4KB_GRP is implemented.
- Table 3-2 for the error record group memory-mapped registers when FEAT_RASSA_16KB_GRP is implemented.
- Table 3-3 for the error record group memory-mapped registers when FEAT_RASSA_64KB_GRP is implemented.
- Table 3-4 for the single error record memory-mapped registers.
- Table 3-5 for the fault injection group memory-mapped registers when FEAT_RASSA_4KB_GRP is implemented.
- Table 3-6 for the fault injection group memory-mapped registers when FEAT_RASSA_16KB_GRP is implemented.
- Table 3-7 for the fault injection group memory-mapped registers when FEAT_RASSA_64KB_GRP is implemented.

Each entry in the Name column links to the register description in RAS register descriptions.

The number of ERRGSR<m> registers is the ceiling of (N÷64). This means that:

- When FEAT_RASSA_4KB_GRP is implemented, there is a single ERRGSR register.
- When FEAT_RASSA_16KB_GRP is implemented, the ERRGSR<m> registers are ERRGSR0 to ERRGSR3.
- When FEAT_RASSA_64KB_GRP is implemented, the ERRGSR<m> registers are ERRGSR0 to ERRGSR13.

Table 3-1 Standard memory-mapped view of group of error records, 4KB page

Name	Type	Size	Description	Offset
ERR<n>FR	RO	64	Error Record Feature Register	0x000+64×n
ERR<n>CTLR	RW	64	Error Record Control Register	0x008+64×n
ERR<n>STATUS	RW	64	Error Record Primary Status Register	0x010+64×n
ERR<n>ADDR	RW	64	Error Record Address Register	0x018+64×n
ERR<n>MISC0	RW	64	Error Record Miscellaneous Register 0	0x020+64×n
ERR<n>MISC1	RW	64	Error Record Miscellaneous Register 1	0x028+64×n
ERR<n>MISC2	RW	64	Error Record Miscellaneous Register 2	0x030+64×n
ERR<n>MISC3	RW	64	Error Record Miscellaneous Register 3	0x038+64×n
ERRIMPDEF<m>	RW	64	IMPLEMENTATION DEFINED Register	0x800+8×m
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x800+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x808+64×n

Name	Type	Size	Description	Offset
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x810+64×n
ERRGSR	RO	64	Error Group Status Register	0xE00
ERRIIDR	RO	32	Error Group Implementation Identification Register	0xE10
ERRACR	RW	64	Access Configuration Register	0xE40
ERRIRQCR<m>	RW	64	Generic Error Interrupt Configuration Register	0xE80+64×m
ERRFHICR0	RW	64	Fault Handling Interrupt Configuration Register 0	0xE80
ERRFHICR1	RW	32	Fault Handling Interrupt Configuration Register 1	0xE88
ERRFHICR2	RW	32	Fault Handling Interrupt Configuration Register 2	0xE8C
ERRERICR0	RW	64	Error Recovery Interrupt Configuration Register 0	0xE90
ERRERICR1	RW	32	Error Recovery Interrupt Configuration Register 1	0xE98
ERRERICR2	RW	32	Error Recovery Interrupt Configuration Register 2	0xE9C
ERRCRICR0	RW	64	Critical Error Interrupt Configuration Register 0	0xEA0
ERRCRICR1	RW	32	Critical Error Interrupt Configuration Register 1	0xEA8
ERRCRICR2	RW	32	Critical Error Interrupt Configuration Register 2	0xEAC
ERRIRQSR	RW	64	Error Interrupt Status Register	0xEF8
ERRDEVAFF	RO	64	Device Affinity Register	0xFA8
ERRDEVARCH	RO	32	Device Architecture Register	0xFBC
ERRDEVID	RO	32	Device Configuration Register	0xFC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0xFD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0xFE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0xFE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0xFE8
ERRPIDR3	RO	32	Peripheral Identification Register 3	0xFEC
ERRCIDR0	RO	32	Component Identification Register 0	0xFF0
ERRCIDR1	RO	32	Component Identification Register 1	0xFF4
ERRCIDR2	RO	32	Component Identification Register 2	0xFF8
ERRCIDR3	RO	32	Component Identification Register 3	0xFFC

Table 3-2 Standard memory-mapped view of group of error records, 16KB page

Name	Type	Size	Description	Offset
ERR<n>FR	RO	64	Error Record Feature Register	0x0000+64×n
ERR<n>CTLR	RW	64	Error Record Control Register	0x0008+64×n
ERR<n>STATUS	RW	64	Error Record Primary Status Register	0x0010+64×n
ERR<n>ADDR	RW	64	Error Record Address Register	0x0018+64×n
ERR<n>MISC0	RW	64	Error Record Miscellaneous Register 0	0x0020+64×n
ERR<n>MISC1	RW	64	Error Record Miscellaneous Register 1	0x0028+64×n
ERR<n>MISC2	RW	64	Error Record Miscellaneous Register 2	0x0030+64×n

Name	Type	Size	Description	Offset
ERR<n>MISC3	RW	64	Error Record Miscellaneous Register 3	0x0038+64×n
ERRIMPDEF<m>	RW	64	IMPLEMENTATION DEFINED Register	0x2000+8×m
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x2000+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x2008+64×n
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x2010+64×n
ERRGSR<m>	RO	64	Error Group Status Register	0x3800+8×m
ERRIIDR	RO	32	Error Group Implementation Identification Register	0x3E10
ERRACR	RW	64	Access Configuration Register	0x3E40
ERRIRQCR<m>	RW	64	Generic Error Interrupt Configuration Register	0x3E80+64×m
ERRFHICR0	RW	64	Fault Handling Interrupt Configuration Register 0	0x3E80
ERRFHICR1	RW	32	Fault Handling Interrupt Configuration Register 1	0x3E88
ERRFHICR2	RW	32	Fault Handling Interrupt Configuration Register 2	0x3E8C
ERRERICR0	RW	64	Error Recovery Interrupt Configuration Register 0	0x3E90
ERRERICR1	RW	32	Error Recovery Interrupt Configuration Register 1	0x3E98
ERRERICR2	RW	32	Error Recovery Interrupt Configuration Register 2	0x3E9C
ERRCRICR0	RW	64	Critical Error Interrupt Configuration Register 0	0x3EA0
ERRCRICR1	RW	32	Critical Error Interrupt Configuration Register 1	0x3EA8
ERRCRICR2	RW	32	Critical Error Interrupt Configuration Register 2	0x3EAC
ERRIRQSR	RW	64	Error Interrupt Status Register	0x3EF8
ERRDEVAFF	RO	64	Device Affinity Register	0x3FA8
ERRDEVARCH	RO	32	Device Architecture Register	0x3FBC
ERRDEVID	RO	32	Device Configuration Register	0x3FC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0x3FD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0x3FE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0x3FE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0x3FE8
ERRPIDR3	RO	32	Peripheral Identification Register 3	0x3FEC
ERRCIDR0	RO	32	Component Identification Register 0	0x3FF0
ERRCIDR1	RO	32	Component Identification Register 1	0x3FF4
ERRCIDR2	RO	32	Component Identification Register 2	0x3FF8
ERRCIDR3	RO	32	Component Identification Register 3	0x3FFC

Table 3-3 Standard memory-mapped view of group of error records, 64KB page

Name	Type	Size	Description	Offset
ERR<n>FR	RO	64	Error Record Feature Register	0x0000+64×n
ERR<n>CTLR	RW	64	Error Record Control Register	0x0008+64×n
ERR<n>STATUS	RW	64	Error Record Primary Status Register	0x0010+64×n

Name	Type	Size	Description	Offset
ERR<n>ADDR	RW	64	Error Record Address Register	0x0018+64×n
ERR<n>MISC0	RW	64	Error Record Miscellaneous Register 0	0x0020+64×n
ERR<n>MISC1	RW	64	Error Record Miscellaneous Register 1	0x0028+64×n
ERR<n>MISC2	RW	64	Error Record Miscellaneous Register 2	0x0030+64×n
ERR<n>MISC3	RW	64	Error Record Miscellaneous Register 3	0x0038+64×n
ERRIMPDEF<m>	RW	64	IMPLEMENTATION DEFINED Register	0x8000+8×m
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x8000+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x8008+64×n
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x8010+64×n
ERRGSR<m>	RO	64	Error Group Status Register	0xE000+8×m
ERRIIDR	RO	32	Error Group Implementation Identification Register	0xFE10
ERRACR	RW	64	Access Configuration Register	0xFE40
ERRIRQCR<m>	RW	64	Generic Error Interrupt Configuration Register	0xFE80+64×m
ERRFHICR0	RW	64	Fault Handling Interrupt Configuration Register 0	0xFE80
ERRFHICR1	RW	32	Fault Handling Interrupt Configuration Register 1	0xFE88
ERRFHICR2	RW	32	Fault Handling Interrupt Configuration Register 2	0xFE8C
ERRERICR0	RW	64	Error Recovery Interrupt Configuration Register 0	0xFE90
ERRERICR1	RW	32	Error Recovery Interrupt Configuration Register 1	0xFE98
ERRERICR2	RW	32	Error Recovery Interrupt Configuration Register 2	0xFE9C
ERRCRICR0	RW	64	Critical Error Interrupt Configuration Register 0	0xFEA0
ERRCRICR1	RW	32	Critical Error Interrupt Configuration Register 1	0xFEA8
ERRCRICR2	RW	32	Critical Error Interrupt Configuration Register 2	0xFEAC
ERRIRQSR	RW	64	Error Interrupt Status Register	0xFEF8
ERRDEVAFF	RO	64	Device Affinity Register	0xFFA8
ERRDEVARCH	RO	32	Device Architecture Register	0xFFBC
ERRDEVID	RO	32	Device Configuration Register	0xFFC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0xFFD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0xFFE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0xFFE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0xFFE8
ERRPIDR3	RO	32	Peripheral Identification Register 3	0xFFEC
ERRCIDR0	RO	32	Component Identification Register 0	0xFFF0
ERRCIDR1	RO	32	Component Identification Register 1	0xFFF4
ERRCIDR2	RO	32	Component Identification Register 2	0xFFF8
ERRCIDR3	RO	32	Component Identification Register 3	0xFFFC

Table 3-4 RAS, single error record, memory-mapped register map

Name	Type	Size	Description	Offset
ERR<n>FR	RO	64	Error Record Feature Register	0x000
ERR<n>CTLR	RW	64	Error Record Control Register	0x008
ERR<n>STATUS	RW	64	Error Record Primary Status Register	0x010
ERR<n>ADDR	RW	64	Error Record Address Register	0x018
ERR<n>MISC0	RW	64	Error Record Miscellaneous Register 0	0x020
ERR<n>MISC1	RW	64	Error Record Miscellaneous Register 1	0x028
ERR<n>MISC2	RW	64	Error Record Miscellaneous Register 2	0x030
ERR<n>MISC3	RW	64	Error Record Miscellaneous Register 3	0x038

Table 3-5 RAS, fault injection group, memory-mapped register map, 4KB page

Name	Type	Size	Description	Offset
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x000+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x008+64×n
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x010+64×n
ERRIIDR	RO	32	Implementation Identification Register	0xE10
ERRACR	RW	64	Access Configuration Register	0xE40
ERRDEVAFF	RO	64	Device Affinity Register	0xFA8
ERRDEVARCH	RO	32	Device Architecture Register	0xFBC
ERRDEVID	RO	32	Device Configuration Register	0xFC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0xFD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0xFE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0xFE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0xFE8
ERRPIDR3	RO	32	Peripheral Identification Register 3	0xFEC
ERRCIDR0	RO	32	Component Identification Register 0	0xFF0
ERRCIDR1	RO	32	Component Identification Register 1	0xFF4
ERRCIDR2	RO	32	Component Identification Register 2	0xFF8
ERRCIDR3	RO	32	Component Identification Register 3	0xFFC

Table 3-6 RAS, fault injection group, memory-mapped register map, 16KB page

Name	Type	Size	Description	Offset
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x0000+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x0008+64×n
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x0010+64×n
ERRIIDR	RO	32	Implementation Identification Register	0x3E10
ERRACR	RW	64	Access Configuration Register	0x3E40
ERRDEVAFF	RO	64	Device Affinity Register	0x3FA8
ERRDEVARCH	RO	32	Device Architecture Register	0x3FBC
ERRDEVID	RO	32	Device Configuration Register	0x3FC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0x3FD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0x3FE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0x3FE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0x3FE8
ERRPIDR3	RO	32	Peripheral Identification Register 3	0x3FEC
ERRCIDR0	RO	32	Component Identification Register 0	0x3FF0
ERRCIDR1	RO	32	Component Identification Register 1	0x3FF4
ERRCIDR2	RO	32	Component Identification Register 2	0x3FF8
ERRCIDR3	RO	32	Component Identification Register 3	0x3FFC

Table 3-7 RAS, fault injection group, memory-mapped register map, 64KB page

Name	Type	Size	Description	Offset
ERR<n>PFGF	RO	64	Error Record Pseudo-fault Generation Feature Register	0x0000+64×n
ERR<n>PFGCTL	RW	64	Error Record Pseudo-fault Generation Control Register	0x0008+64×n
ERR<n>PFGCDN	RW	64	Error Record Pseudo-fault Generation Countdown Register	0x0010+64×n
ERRIIDR	RO	32	Implementation Identification Register	0xFE10
ERRACR	RW	64	Access Configuration Register	0xFE40
ERRDEVAFF	RO	64	Device Affinity Register	0xFFA8
ERRDEVARCH	RO	32	Device Architecture Register	0xFFBC
ERRDEVID	RO	32	Device Configuration Register	0xFFC8
ERRPIDR4	RO	32	Peripheral Identification Register 4	0xFFD0
ERRPIDR0	RO	32	Peripheral Identification Register 0	0xFFE0
ERRPIDR1	RO	32	Peripheral Identification Register 1	0xFFE4
ERRPIDR2	RO	32	Peripheral Identification Register 2	0xFFE8

Name	Type	Size	Description	Offset
ERRPIDR3	RO	32	Peripheral Identification Register 3	0xFFEC
ERRCIDR0	RO	32	Component Identification Register 0	0xFFF0
ERRCIDR1	RO	32	Component Identification Register 1	0xFFF4
ERRCIDR2	RO	32	Component Identification Register 2	0xFFF8
ERRCIDR3	RO	32	Component Identification Register 3	0xFFFC

3.2 RAS register descriptions

This section describes the RAS registers.

3.2.1 ERRACR, Access Configuration Register

The ERRACR characteristics are:

Purpose

Controls visibility of error records.

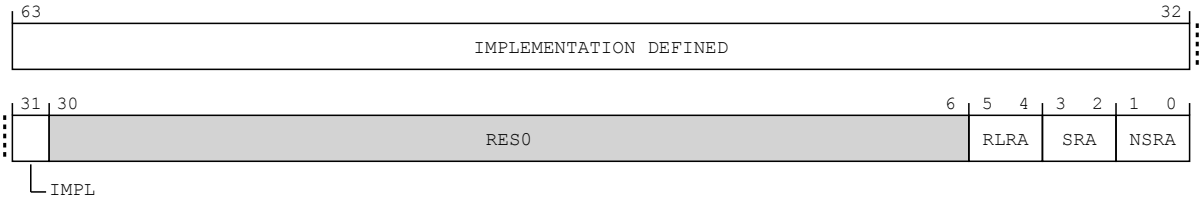
Configuration

This register is present only when (Root state is implemented or Secure state is implemented) and an implementation implements ERRACR. Otherwise, direct accesses to ERRACR are RES0.

Attributes

ERRACR is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED access control bits.

IMPL, bit [31]

IMPL	Meaning
0b1	Indicates ERRACR is present.

Access to this field is RAO/WI.

Bits [30:6]

Reserved, RES0.

RLRA, bits [5:4]

When FEAT_RME is implemented and the error record group allows configuration of Secure and Realm register accesses:

Realm Restricted Access. Controls Realm access to error records and interrupt configuration registers in the error record group.

RLRA	Meaning
0b00	Realm access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Realm accesses.
0b01	Realm read access is enabled. Realm writes are ignored.
0b11	Realm read/write access is allowed. If the error record group supports MSIs, generated MSIs are always Non-secure.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Realm access to error records is disabled, a Realm read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Realm access is fully or partially disabled, the effect on Realm accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Realm access to error records is enabled from reset.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RAZ/WI.

SRA, bits [3:2]

When Secure state is implemented, FEAT_RME is implemented, and the error record group allows configuration of Secure and Realm register accesses:

Secure Restricted Access. Controls Secure access to error records and interrupt configuration registers in the error record group.

SRA	Meaning
0b00	Secure access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Secure accesses.
0b01	Secure read access is enabled. Secure writes are ignored.
0b11	Secure read/write access is allowed.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Secure access to error records is disabled, a Secure read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Secure access is fully or partially disabled, the effect on Secure accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Secure access to error records is enabled from reset.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RAZ/WI.

NSRA, bits [1:0]

Non-secure Restricted Access. Controls Non-secure access to error records and interrupt configuration registers in the error record group.

NSRA	Meaning
0b00	Non-secure access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Non-secure accesses.
0b01	Non-secure read access is enabled. Non-secure writes are ignored.
0b11	Non-secure read/write access is allowed. If the error record group supports MSIs, generated MSIs are always Non-secure.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Non-secure access to error records is disabled, a Non-secure read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Non-secure access is fully or partially disabled, the effect on Non-secure accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Non-secure access to error records is enabled from reset.

If FEAT_RME is implemented and ERRACR.{RLRA, SRA} are not implemented, then ERRACR.NSRA applies to all Security states other than Root.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing ERRACR

This section shows the offset of ERRACR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRACR.

ERRACR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xE40`	ERRACR

Accessible as follows:

- When (FEAT_RME is implemented and an access is not Root) or an access is Non-secure, accesses to this register are RAZ/WI.
- Otherwise, accesses to this register are RW.

3.2.2 ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

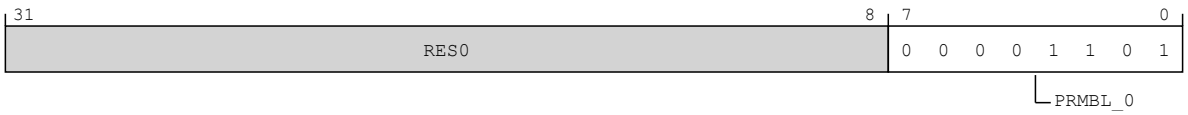
ERRCIDR0 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRCIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.

Reads as 0x0D

Access to this field is RO.

Accessing ERRCIDR0

This section shows the offset of ERRCIDR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCIDR0.

ERRCIDR0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFF0	ERRCIDR0

Accesses to this register are RO.

3.2.3 ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

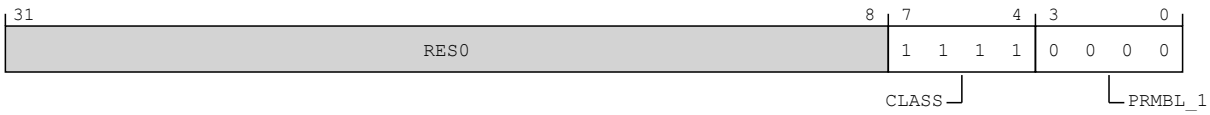
ERRCIDR1 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRCIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1111	Generic peripheral with IMPLEMENTATION DEFINED register layout.

Other values are defined by the CoreSight Architecture.

This field reads as 0xF.

Access to this field is RO.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000

Access to this field is RO.

Accessing ERRCIDR1

This section shows the offset of ERRCIDR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCIDR1.

ERRCIDR1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFF4	ERRCIDR1

Accesses to this register are RO.

3.2.4 ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

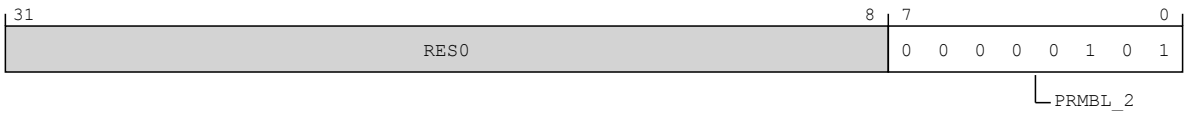
ERRCIDR2 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRCIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.

Reads as `0x05`

Access to this field is RO.

Accessing ERRCIDR2

This section shows the offset of ERRCIDR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCIDR2.

ERRCIDR2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xFF8`	ERRCIDR2

Accesses to this register are RO.

3.2.5 ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

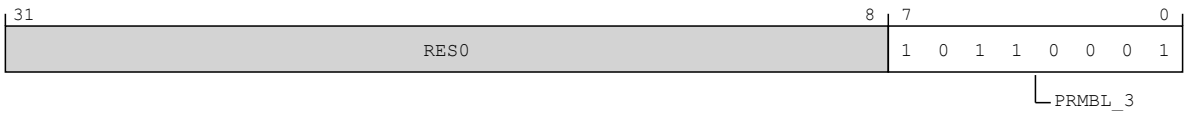
ERRCIDR3 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRCIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.

Reads as `0xB1`

Access to this field is RO.

Accessing ERRCIDR3

This section shows the offset of ERRCIDR3 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCIDR3.

ERRCIDR3 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xFFC`	ERRCIDR3

Accesses to this register are RO.

3.2.6 ERRCRICR0, Critical Error Interrupt Configuration Register 0

The ERRCRICR0 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

ERRCRICR0 is implemented only as part of a memory-mapped group of error records.

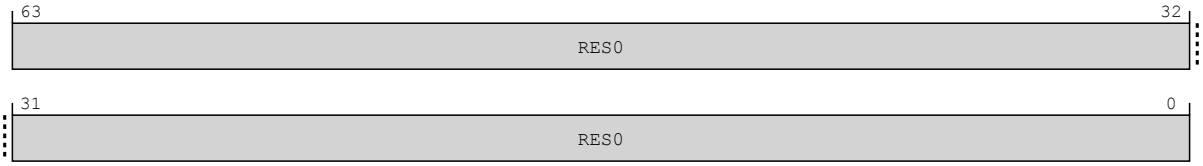
This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR0 are RES0.

Attributes

ERRCRICR0 is a 64-bit register.

Field descriptions

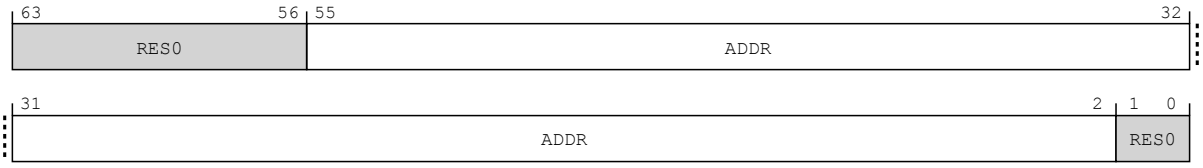
When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRCRICR0.ADDR « 2) is the address that the component writes to when signaling the Critical Error Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

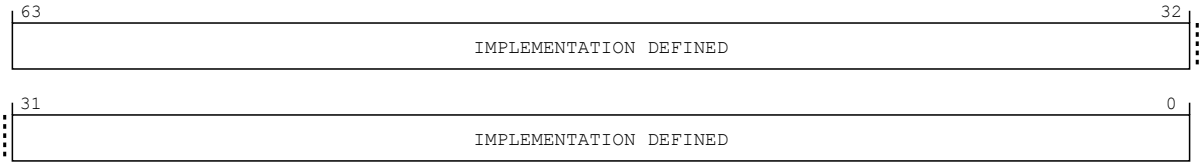
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR0 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRCRICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCRICR0.

ERRCRICR0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xEA0	ERRCRICR0

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.7 ERRCRICR1, Critical Error Interrupt Configuration Register 1

The ERRCRICR1 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

ERRCRICR1 is implemented only as part of a memory-mapped group of error records.

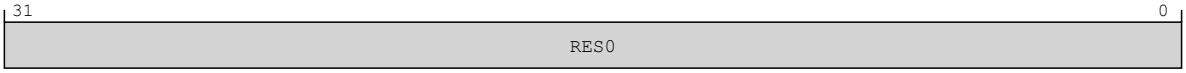
This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR1 are RES0.

Attributes

ERRCRICR1 is a 32-bit register.

Field descriptions

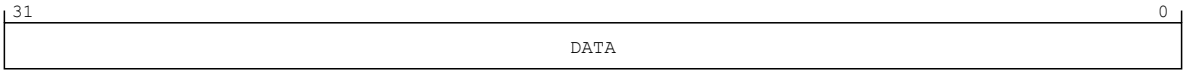
When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR1 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRCRICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCRICR1.

ERRCRICR1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xEA8`	ERRCRICR1

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.8 ERRCRICR2, Critical Error Interrupt Configuration Register 2

The ERRCRICR2 characteristics are:

Purpose

Critical Error Interrupt control and configuration register.

Configuration

ERRCRICR2 is implemented only as part of a memory-mapped group of error records.

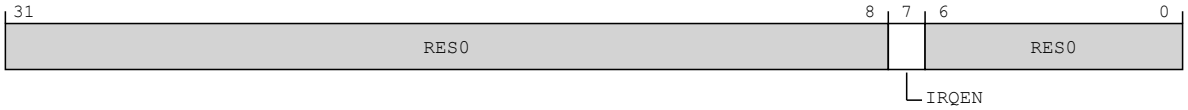
This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR2 are RES0.

Attributes

ERRCRICR2 is a 32-bit register.

Field descriptions

When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

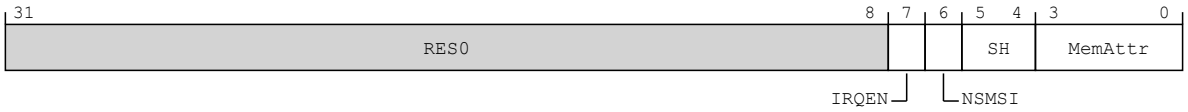
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is RO.
- When an access is Realm, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRCRICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.

MemAttr	Meaning
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR2 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRCRICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRCRICR2.

ERRCRICR2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xEAC	ERRCRICR2

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.9 ERRDEVAFF, Device Affinity Register

The ERRDEVAFF characteristics are:

Purpose

For a group of error records that has affinity with a single PE or a group of PEs, ERRDEVAFF is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the group of error records has affinity with a single PE, the affinity level is 0, then ERRDEVAFF reads the same value as MPIDR_EL1, and ERRDEVAFF.F0V reads-as-one to indicate affinity level 0.
- If the group of error records has affinity with a group of PEs, the affinity level is 1, 2, or 3, then parts of ERRDEVAFF reads the same value as parts of MPIDR_EL1, and the rest of ERRDEVAFF indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1, then all of the following are true:

- All the PEs in the group have the same values in MPIDR_EL1.{Aff3,Aff2}, and these values are equal to ERRDEVAFF.{Aff3,Aff2}.
- ERRDEVAFF.Aff1 is nonzero and not `0x80`, and ERRDEVAFF.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the group of error records has affinity with is indicated by the least-significant set bit in ERRDEVAFF.Aff1. In this example, if ERRDEVAFF.Aff1[2:0] is `0b100`, then the group of error records has affinity with the up-to 8 PEs that have MPIDR_EL1.Aff1[7:3] == ERRDEVAFF.Aff1[7:3].

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that ERRDEVAFF is read before system firmware has configured the group of error records or the PE or group of PEs that the group of error records has affinity with. When this is the case, ERRDEVAFF reads as zero.

If RAS System Architecture v1.1 is not implemented, then ERRDEVAFF can only describe a group of error records that is affine with a single PE or all the PEs at an affinity level.

Configuration

ERRDEVAFF is implemented only as part of a memory-mapped group of error records.

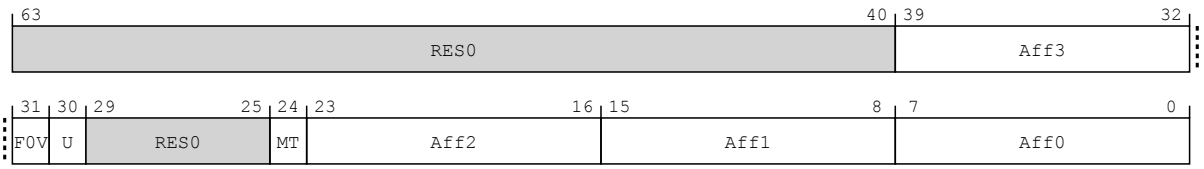
Arm deprecates use of this register by software

This register is present only when the group of error records has affinity with a PE or cluster of PEs and an implementation implements ERRDEVAFF. Otherwise, direct accesses to ERRDEVAFF are RES0.

Attributes

ERRDEVAFF is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

F0V, bit [31]

Indicates that the ERRDEVAFF.Aff0 field is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F0V	Meaning
0b0	ERRDEVAFF.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
0b1	ERRDEVAFF.Aff0 is valid, and the PE affinity is at level 0.

Access to this field is RO.

U, bit [30]

When ERRDEVAFF.F0V == '1':

Uniprocessor. The MPIDR_EL1.U field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

When ERRDEVAFF.F0V == '1':

Multithreaded. The MPIDR_EL1.MT field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Aff2, bits [23:16]

When !IsZero([ERRDEVAFF.Aff1, ERRDEVAFF.Aff0, ERRDEVAFF.F0V]):

PE affinity level 2. The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

PE affinity level 2. Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff2	Meaning
0bxxxxxxx1	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 3.

Access to this field is RO.

Aff1, bits [15:8]

When !IsZero([ERRDEVAFF.Aff0, ERRDEVAFF.F0V]):

PE affinity level 1. The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

PE affinity level 1. Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0x00	PE affinity is above level 2 or a subset of level 2.
0bxxxxxxx1	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.

Aff1	Meaning
0bx1000000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 2.

Access to this field is RO.

Aff0, bits [7:0]

When ERRDEVAFF.F0V == '1':

PE affinity level 0. The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

PE affinity level 0. Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0x00	PE affinity is above level 1 or a subset of level 1.
0bxxxxxxx1	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 1.

Access to this field is RO.

Accessing ERRDEVAFF

This section shows the offset of ERRDEVAFF when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRDEVAFF.

ERRDEVAFF can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFA8	ERRDEVAFF

Accesses to this register are RO.

3.2.10 ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

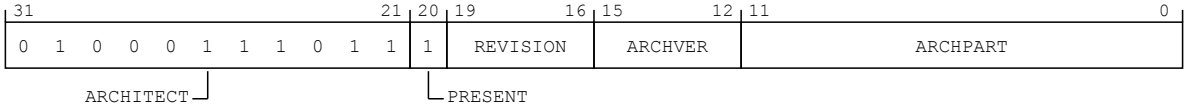
Configuration

ERRDEVARCH is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For RAS, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`

Access to this field is RO.

PRESENT, bit [20]

DEVARCH present. Indicates that the ERRDEVARCH register is present.

Reads as `0b1`

Access to this field is RO.

REVISION, bits [19:16]

When UInt(ERRDEVARCH.ARCHPART) == `0xA00` and ERRDEVARCH.ARCHVER == '0000':

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
`0b0000`	RAS System Architecture, error record group v1.0.
`0b0001`	RAS System Architecture, error record group v1.1. As `0b0000` and also: • Simplifies ERR<n>STATUS. • Adds support for additional ERR<n>MISC<m> registers. • Adds support for the optional RAS Timestamp Extension. • Adds support for the optional Common Fault Injection Model Extension.

All other values are reserved.

Access to this field is RO.

When UInt(ERRDEVARCH.ARCHPART) == `0xA00` and ERRDEVARCH.ARCHVER == '0001':

Revision. Defines the architecture revision of the component.

REVISION	Meaning
`0b0000`	RAS System Architecture, error record group v2.0.

All other values are reserved.

Access to this field is RO.

When UInt(ERRDEVARCH.ARCHPART) == `0xA08` and ERRDEVARCH.ARCHVER == '0000':

Revision. Defines the architecture revision of the component.

REVISION	Meaning
`0b0000`	RAS System Architecture, fault injection group v1.0.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

ARCHVER, bits [15:12]

When UInt(ERRDEVARCH.ARCHPART) == `0xA00`:

Architecture Version. Defines the architecture version of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHVER	Meaning
`0b0000`	RAS System Architecture, error record group v1.
`0b0001`	RAS System Architecture, error record group v2. As `0b0000` and also: • Adds an optional access control register, ERRACR. • Adds an optional control for disabling error counters. • Adds optional fault handling interrupt controls for Deferred errors. • Adds support for continuation and proxy error records. • Adds support for implementing Common Fault Injection Mechanism registers in a separate page from the error record registers. • Adds support for simple interrupt control registers. • Defines fields in ERRDEVID that describe these properties.

All other values are reserved.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHVER is ERRDEVARCH.ARCHID[15:12].

Access to this field is RO.

When UInt(ERRDEVARCH.ARCHPART) == `0xA08`:

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
`0b0000`	RAS System Architecture, fault injection group v1.

All other values are reserved.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHVER is ERRDEVARCH.ARCHID[15:12].

Access to this field is RO.

Otherwise:

Reserved, RES0.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHPART	Meaning
`0xA00`	RAS System Architecture, error record group.
`0xA08`	RAS System Architecture, fault injection group.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHPART is ERRDEVARCH.ARCHID[11:0].

Access to this field is RO.

Accessing ERRDEVARCH

This section shows the offset of ERRDEVARCH when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRDEVARCH.

ERRDEVARCH can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xFBC`	ERRDEVARCH

Accesses to this register are RO.

3.2.11 ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

Purpose

Provides discovery information for the component.

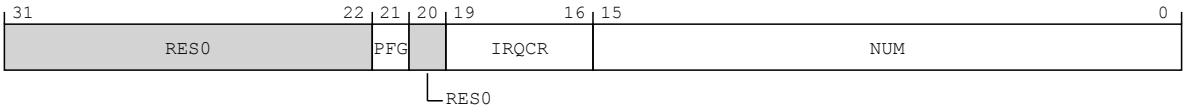
Configuration

ERRDEVID is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVID is a 32-bit register.

Field descriptions



Bits [31:22]

Reserved, RES0.

PFG, bit [21]

When RAS System Architecture v2 is implemented:

Common Fault Injection Mechanism. Describes whether any Common Fault Injection Mechanism registers are implemented in the same page as this register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PFG	Meaning
0b0	Any Common Fault Injection Mechanism registers are implemented in the same page as this register.
0b1	Any Common Fault Injection Mechanism registers are implemented in a separate fault injection group page.

Note

The value of this field does not indicate that any Common Fault Injection Mechanism registers are implemented by the nodes in this error record group. Software must use ERR<n>FR to discover whether each node implements Common Fault Injection Mechanism registers.

Accessing this field has the following behavior:

- When ERRDEVID is part of a fault injection group, access to this field is RAZ/WI.
- Otherwise, access to this field is RO.

Otherwise:

Reserved, RAZ.

Bit [20]

Reserved, RES0.

IRQCR, bits [19:16]

Interrupt Control registers. Describes whether the interrupt control registers are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IRQCR	Meaning
0b0000	It is IMPLEMENTATION DEFINED whether any interrupt control registers are implemented.
0b0001	An IMPLEMENTATION DEFINED form of interrupt control registers are implemented.
0b0010	The recommended layout form of interrupt control registers are implemented, for simple interrupts.
0b0011	The recommended layout form of interrupt control registers are implemented, for message-signaled interrupts.
0b1111	Interrupt control registers are not implemented.

All other values are reserved.

Accessing this field has the following behavior:

- When ERRDEVID is part of a fault injection group, access to this field is RAZ/WI.
- Otherwise, access to this field is RO.

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 has an IMPLEMENTATION DEFINED value.

This manual describes a group of error records accessed via a standard 4KB memory-mapped peripheral. For a 4KB peripheral, up to 24 error records can be accessed if the Common Fault Injection Model is implemented, and up to 56 otherwise.

Access to this field is RO.

Accessing ERRDEVID

This section shows the offset of ERRDEVID when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRDEVID.

ERRDEVID can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFC8	ERRDEVID

Accesses to this register are RO.

3.2.12 ERRERICR0, Error Recovery Interrupt Configuration Register 0

The ERRERICR0 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

ERRERICR0 is implemented only as part of a memory-mapped group of error records.

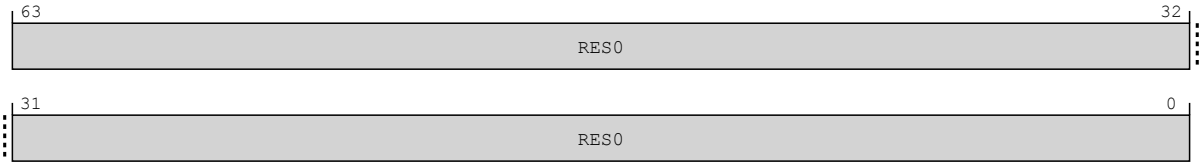
This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR0 are RES0.

Attributes

ERRERICR0 is a 64-bit register.

Field descriptions

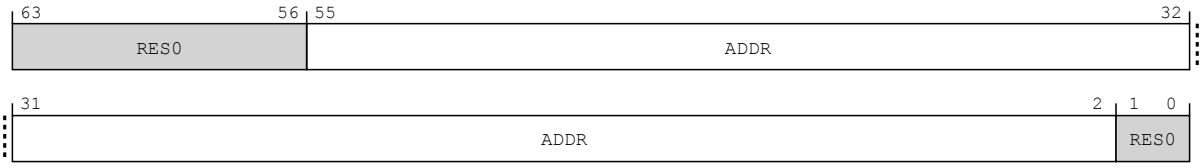
When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRERICR0.ADDR « 2) is the address that the component writes to when signaling the Error Recovery Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

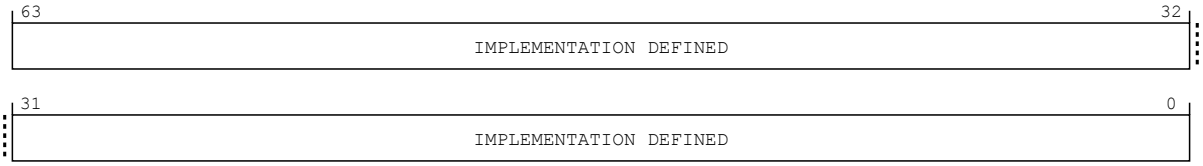
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR0 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRERICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRERICR0.

ERRERICR0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE90	ERRERICR0

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.13 ERRERICR1, Error Recovery Interrupt Configuration Register 1

The ERRERICR1 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

ERRERICR1 is implemented only as part of a memory-mapped group of error records.

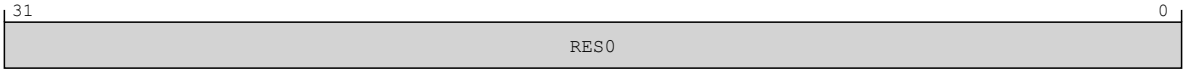
This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR1 are RES0.

Attributes

ERRERICR1 is a 32-bit register.

Field descriptions

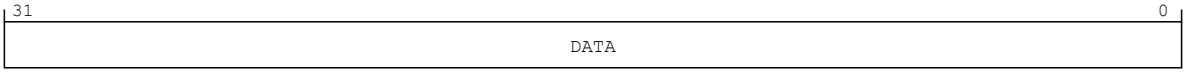
When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR1 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRERICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRERICR1.

ERRERICR1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE98	ERRERICR1

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.14 ERRERICR2, Error Recovery Interrupt Configuration Register 2

The ERRERICR2 characteristics are:

Purpose

Error Recovery Interrupt control and configuration register.

Configuration

ERRERICR2 is implemented only as part of a memory-mapped group of error records.

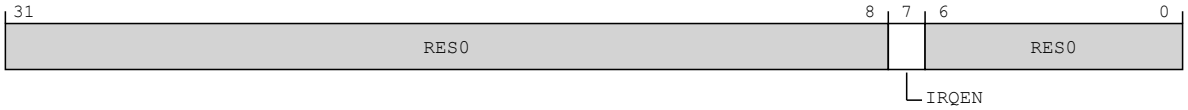
This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR2 are RES0.

Attributes

ERRERICR2 is a 32-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

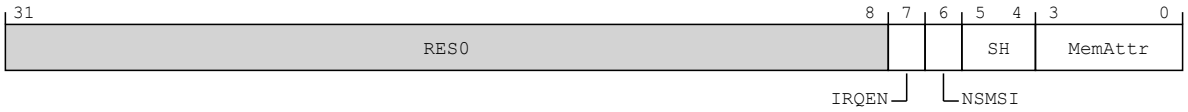
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is RO.
- When an access is Realm, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRERICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.

MemAttr	Meaning
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR2 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRERICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRERICR2.

ERRERICR2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE9C	ERRERICR2

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.15 ERRFHICR0, Fault Handling Interrupt Configuration Register 0

The ERRFHICR0 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

ERRFHICR0 is implemented only as part of a memory-mapped group of error records.

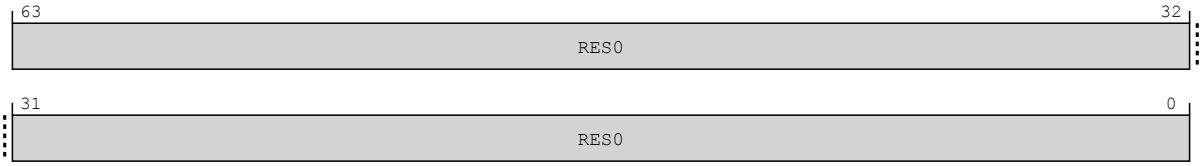
This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR0 are RES0.

Attributes

ERRFHICR0 is a 64-bit register.

Field descriptions

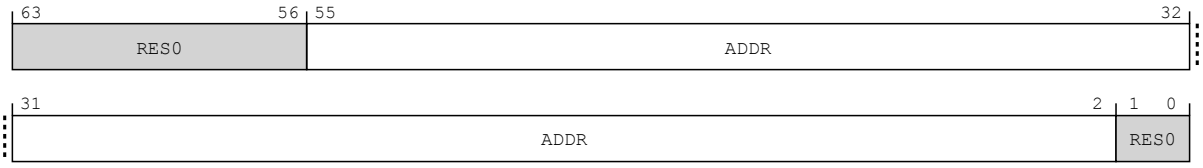
When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRFHICR0.ADDR « 2) is the address that the component writes to when signaling the Fault Handling Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

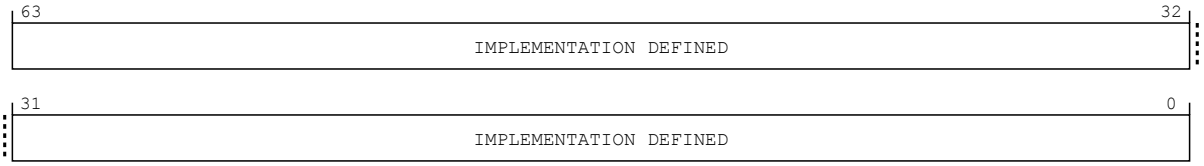
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR0 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRFHICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRFHICR0.

ERRFHICR0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE80	ERRFHICR0

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.16 ERRFHICR1, Fault Handling Interrupt Configuration Register 1

The ERRFHICR1 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

ERRFHICR1 is implemented only as part of a memory-mapped group of error records.

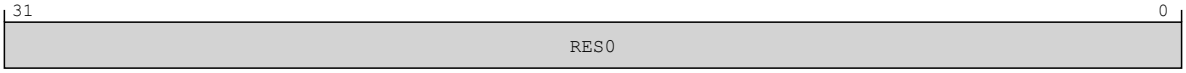
This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR1 are RES0.

Attributes

ERRFHICR1 is a 32-bit register.

Field descriptions

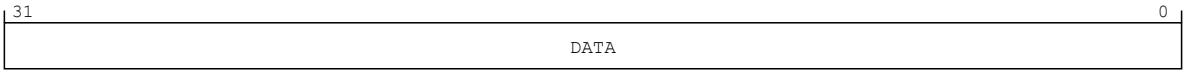
When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR1 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRFHICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRFHICR1.

ERRFHICR1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE88	ERRFHICR1

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.17 ERRFHICR2, Fault Handling Interrupt Configuration Register 2

The ERRFHICR2 characteristics are:

Purpose

Fault Handling Interrupt control and configuration register.

Configuration

ERRFHICR2 is implemented only as part of a memory-mapped group of error records.

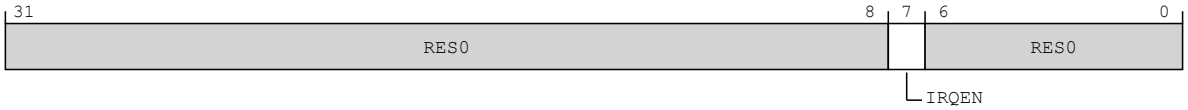
This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR2 are RES0.

Attributes

ERRFHICR2 is a 32-bit register.

Field descriptions

When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

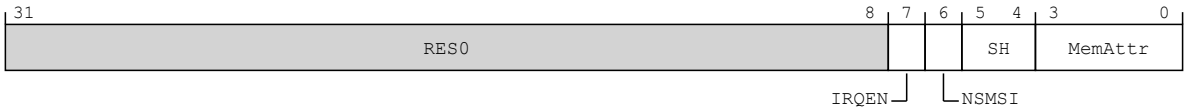
The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is RO.
- When an access is Realm, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRFHICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.

MemAttr	Meaning
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR2 are IMPLEMENTATION DEFINED.

This section shows the offset of ERRFHICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRFHICR2.

ERRFHICR2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xE8C	ERRFHICR2

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.18 ERRGSR<m>, Error Group <n> Status Register, m = 0 - 13

The ERRGSR<m> characteristics are:

Purpose

Shows the status for the records in the group.

Configuration

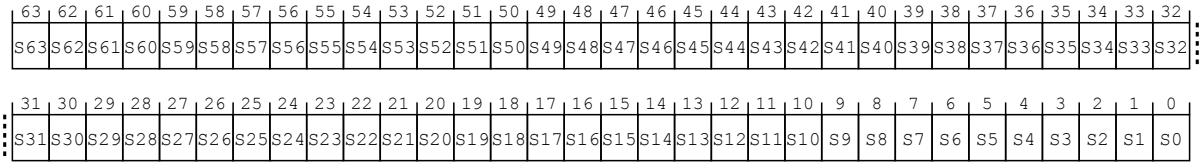
ERRGSR is implemented only as part of a memory-mapped group of error records.

If FEAT_RASSA_4KB_GRP is implemented, then a single ERRGSR register is implemented.

Attributes

ERRGSR<m> is a 64-bit register.

Field descriptions



S<n> , bits [n], for n = 63 to 0

If FEAT_RASSA_4KB_GRP is implemented, the status for error record <n>. A read-only copy of ERR<n>STATUS.V.

If FEAT_RASSA_16KB_GRP is implemented or FEAT_RASSA_64KB_GRP is implemented, the status for error record <m×64+n>. A read-only copy of ERR<m×64+n>STATUS.V.

S<n>	Meaning
0b0	No error.
0b1	One or more errors.

Accessing this field has the following behavior:

- Access to this field is RES0 if all of the following are true:
  - FEAT_RASSA_4KB_GRP is implemented
  - n >= 56
- Access to this field is RES0 if all of the following are true:
  - FEAT_RASSA_4KB_GRP is implemented
  - the Common Fault Injection Model is implemented by any error record in the group
  - n >= 24
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_RASSA_4KB_GRP is implemented
  - error record n is not implemented
- Access to this field is RAZ/WI if all of the following are true:
  - Any of the following are true:
    - FEAT_RASSA_16KB_GRP is implemented
    - FEAT_RASSA_64KB_GRP is implemented

    - error record (m * 64) + n is not implemented
- Access to this field is RAZ/WI if all of the following are true:
    - FEAT_RASSAv2 is not implemented
    - error record n does not support this type of reporting
- Otherwise, access to this field is RO.

Accessing ERRGSR<m>

This section shows the offset of ERRGSR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRGSR<n>.

ERRGSR<m> can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xE00 + (64 * m)`	ERRGSR<m>

Accesses to this register are RO.

3.2.19 ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

Purpose

Defines the implementer of the component.

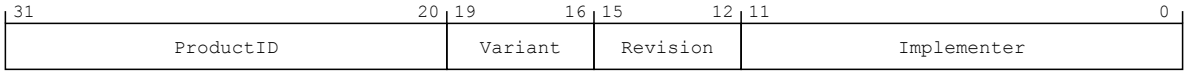
Configuration

This register is present only when RAS System Architecture v1p1 is implemented. Otherwise, direct accesses to ERRIIDR are RES0.

Attributes

ERRIIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

This field has an IMPLEMENTATION DEFINED value.

If ERRPIDR0 and ERRPIDR1 are implemented, ERRPIDR0.PART_0 matches bits [7:0] of ERRIIDR.ProductID and ERRPIDR1.PART_1 matches bits [11:8] of ERRIIDR.ProductID.

Access to this field is RO.

Variant, bits [19:16]

Component major revision.

This field distinguishes product variants or major revisions of the product.

This field has an IMPLEMENTATION DEFINED value.

If ERRPIDR2 is implemented, ERRPIDR2.REVISION matches ERRIIDR.Variant.

Access to this field is RO.

Revision, bits [15:12]

Component minor revision.

This field distinguishes minor revisions of the product.

This field has an IMPLEMENTATION DEFINED value.

If ERRPIDR3 is implemented, ERRPIDR3.REVAND matches ERRIIDR.Revision.

Access to this field is RO.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the RAS component.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for ERRIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

This field has an IMPLEMENTATION DEFINED value.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If ERRPIDR4 is implemented, ERRPIDR4.DES_2 matches bits [11:8] of this field.

If ERRPIDR2 is implemented, ERRPIDR2.DES_1 matches bits [6:4] of this field.

If ERRPIDR1 is implemented, ERRPIDR1.DES_0 matches bits [3:0] of this field.

Access to this field is RO.

Accessing ERRIIDR

This section shows the offset of ERRIIDR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIIDR.

ERRIIDR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xE10`	ERRIIDR

Accesses to this register are RO.

3.2.20 ERRIMPDEF<n>, IMPLEMENTATION DEFINED Register <n>, n = 0 - 191

The ERRIMPDEF<n> characteristics are:

Purpose

IMPLEMENTATION DEFINED RAS extensions.

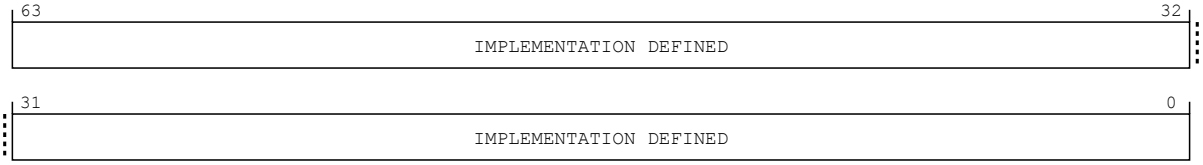
Configuration

This register is present only when ((the Common Fault Injection Model Extension is not implemented && the Fault Injection Group is not implemented) && (((FEAT_RASSA_4KB_GRP is implemented && (UInt(ERRDEVID.NUM) <= 32)) || (FEAT_RASSA_16KB_GRP is implemented && (UInt(ERRDEVID.NUM) <= 128))) || (FEAT_RASSA_64KB_GRP is implemented && (UInt(ERRDEVID.NUM) <= 512)))) && ImpDefBool("IMPLEMENTED_ERRIMPDEF<n>"). Otherwise, direct accesses to ERRIMPDEF<n> are RES0.

Attributes

ERRIMPDEF<n> is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIMPDEF<n>

This section shows the offset of ERRIMPDEF<n> when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIMPDEF<n>.

ERRIMPDEF<n> can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x800 + (8 * n)`	ERRIMPDEF<n>

Accesses to this register are RW.

3.2.21 ERRIRQCR<n>, Generic Error Interrupt Configuration Register <n>, n = 0 - 15

The ERRIRQCR<n> characteristics are:

Purpose

The ERRIRQCR<n> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

The architecture provides a recommended layout for the ERRIRQCR<n> registers. These registers are named:

- ERRFHICR0, ERRFHICR1, and ERRFHICR2 for the fault handling interrupt controls.
- ERRERICR0, ERRERICR1, and ERRERICR2 for the error recovery interrupt controls.
- ERRCRICR0, ERRCRICR1, and ERRCRICR2 for the critical error interrupt controls.
- ERRIRQSR for the status register.

This section describes the generic, IMPLEMENTATION DEFINED, format.

Configuration

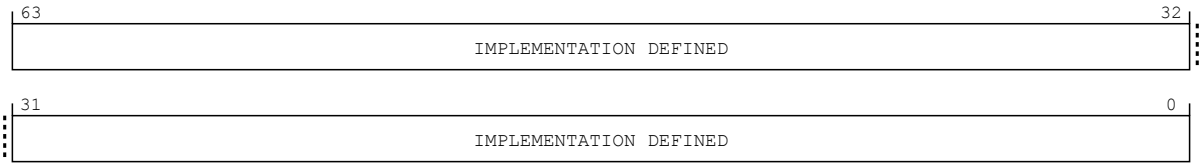
ERRIRQCR<n> is implemented only as part of a memory-mapped group of error records.

This register is present only when the interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQCR<n> are RES0.

Attributes

ERRIRQCR<n> is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

Accessing ERRIRQCR<n>

This section shows the offset of ERRIRQCR<n> when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIRQCR<n>.

ERRIRQCR<n> can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xE80 + (8 * n)`	ERRIRQCR<n>

Accesses to this register are RW.

3.2.22 ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose

Interrupt status register.

Configuration

ERRIRQSR is implemented only as part of a memory-mapped group of error records.

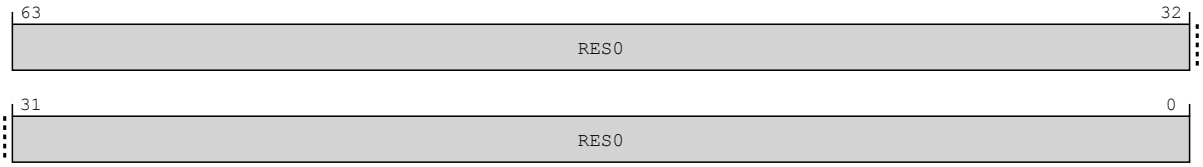
This register is present only when interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQSR are RES0.

Attributes

ERRIRQSR is a 64-bit register.

Field descriptions

When the implementation uses the recommended layout for the ERRIRQCR registers and the implementation uses simple interrupts:

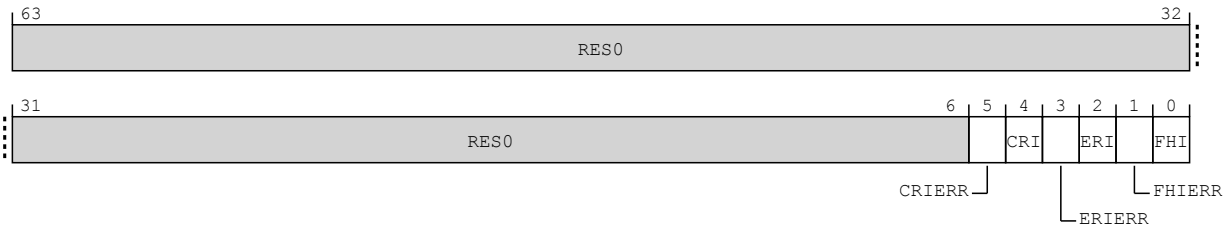


Bits [63:0]

Reserved, RES0.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

When the implementation uses message-signaled interrupts and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:6]

Reserved, RES0.

CRIERR, bit [5]

When the Critical Error Interrupt is implemented:

Critical Error Interrupt Error.

CRIERR	Meaning
0b0	Critical Error Interrupt write has not returned an error since this field was last cleared to zero.
0b1	Critical Error Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

CRI, bit [4]

When the Critical Error Interrupt is implemented:

Critical Error Interrupt write in progress.

CRI	Meaning
0b0	Critical Error Interrupt write not in progress.
0b1	Critical Error Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

This field 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.

Access to this field is RO.

Otherwise:

Reserved, RES0.

ERIERR, bit [3]

When the Error Recovery Interrupt is implemented:

Error Recovery Interrupt Error.

ERIERR	Meaning
0b0	Error Recovery Interrupt write has not returned an error since this field was last cleared to zero.

ERIERR	Meaning
0b1	Error Recovery Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

ERI, bit [2]

When the Error Recovery Interrupt is implemented:

Error Recovery Interrupt write in progress.

ERI	Meaning
0b0	Error Recovery Interrupt write not in progress.
0b1	Error Recovery Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

This field 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.

Access to this field is RO.

Otherwise:

Reserved, RES0.

FHIERR, bit [1]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt Error.

FHIERR	Meaning
0b0	Fault Handling Interrupt write has not returned an error since this field was last cleared to zero.
0b1	Fault Handling Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

FHI, bit [0]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt write in progress.

FHI	Meaning
0b0	Fault Handling Interrupt write not in progress.
0b1	Fault Handling Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

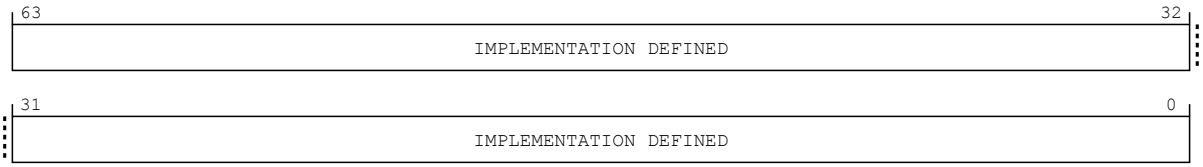
This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.
To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Access to this field is RO.

Otherwise:

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIRQSR

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRIRQSR are IMPLEMENTATION DEFINED.

This section shows the offset of ERRIRQSR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRIRQSR.

ERRIRQSR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xEF8	ERRIRQSR

Accessible as follows:

- When ((the implementation uses message-signaled interrupts && (IsAccessNonSecure(addrdesc) || IsAccessRealm(addrdesc))) && the implementation uses the recommended layout for the ERRIRQCR registers) && ERRIRQSR.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.23 ERR<n>ADDR, Error Record <n> Address Register, n = 0 - 65534

The ERR<n>ADDR characteristics are:

Purpose

If an address is associated with a detected error, then it is written to ERR<n>ADDR when the error is recorded. It is IMPLEMENTATION DEFINED how the recorded address maps to the software-visible physical address. Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

Configuration

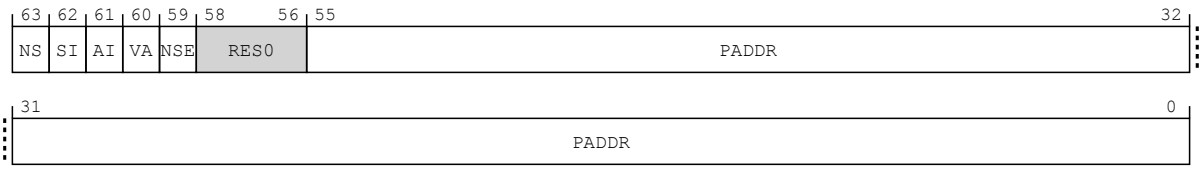
ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

This register is present only when FEAT_RAS is implemented, error record n is implemented, and error record n includes an address associated with an error. Otherwise, direct accesses to ERR<n>ADDR are RES0.

Attributes

ERR<n>ADDR is a 64-bit register.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

Non-secure attribute. With ERR<n>ADDR.NSE, indicates the physical address space of the recorded location.

NS	Meaning
0b0	When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Root address.
0b1	When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Non-secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Realm address.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure attribute.

NS	Meaning
0b0	ERR<n>ADDR.PADDR is a Secure address.
0b1	ERR<n>ADDR.PADDR is a Non-secure address.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SI, bit [62]

When FEAT_RME is implemented:

Secure Incorrect. Indicates whether ERR<n>ADDR.{NS, NSE} are valid.

SI	Meaning
0b0	ERR<n>ADDR.{NS, NSE} are correct. That is, they match the software's view of the physical address space for the recorded location.
0b1	ERR<n>ADDR.{NS, NSE} might not be correct, and might not match the software's view of the physical address space for the recorded location.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Secure Incorrect. Indicates whether ERR<n>ADDR.NS is valid.

SI	Meaning
0b0	ERR<n>ADDR.NS is correct. That is, it matches the software's view of the Non-secure attribute for the recorded location.
0b1	ERR<n>ADDR.NS might not be correct, and might not match the software's view of the Non-secure attribute for the recorded location.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

AI, bit [61]

Address Incorrect. Indicates whether ERR<n>ADDR.PADDR is a valid physical address that is known to match the software's view of the physical address for the recorded location.

AI	Meaning
0b0	ERR<n>ADDR.PADDR is a valid physical address. That is, it matches the software's view of the physical address for the recorded location.
0b1	ERR<n>ADDR.PADDR might not be a valid physical address, and might not match the software's view of the physical address for the recorded location.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

VA, bit [60]

Virtual Address. Indicates whether ERR<n>ADDR.PADDR field is a virtual address.

VA	Meaning
0b0	ERR<n>ADDR.PADDR is not a virtual address.
0b1	ERR<n>ADDR.PADDR is a virtual address.

No context information is provided for the virtual address. When ERR<n>ADDR.VA is recorded as 1, ERR<n>ADDR.{NS, SI, AI} are recorded as {0, 1, 1} and, if FEAT_RME is implemented, ERR<n>ADDR.NSE is recorded as 0.

Support for this field is optional. If this field is not implemented and ERR<n>ADDR.PADDR field is a virtual address, then ERR<n>ADDR.{NS, SI, AI} read as {0, 1, 1} and, if FEAT_RME is implemented, ERR<n>ADDR.NSE reads as 0.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [59]

When FEAT_RME is implemented:

Physical Address Space. Together with ERR<n>ADDR.NS, indicates the address space for ERR<n>ADDR.PADDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, 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.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>ADDR

ERR<n>ADDR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x018 + (64 * n)`	ERR<n>ADDR

Accessible as follows:

- When the node that owns error record n implements the Common Fault Injection Model Extension, ERRPFGF[FirstRecordOfNode(n)].AV == ‘0’, and ERR<n>STATUS.AV == ‘1’, accesses to this register are RO.
- When the node that owns error record n does not implement the Common Fault Injection Model Extension and ERR<n>STATUS.AV == ‘1’, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.24 ERR<n>CTLR, Error Record <n> Control Register, n = 0 - 65534

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 the critical error, error recovery, and fault handling interrupts.
- Enabling in-band error response for uncorrected errors.

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

Configuration

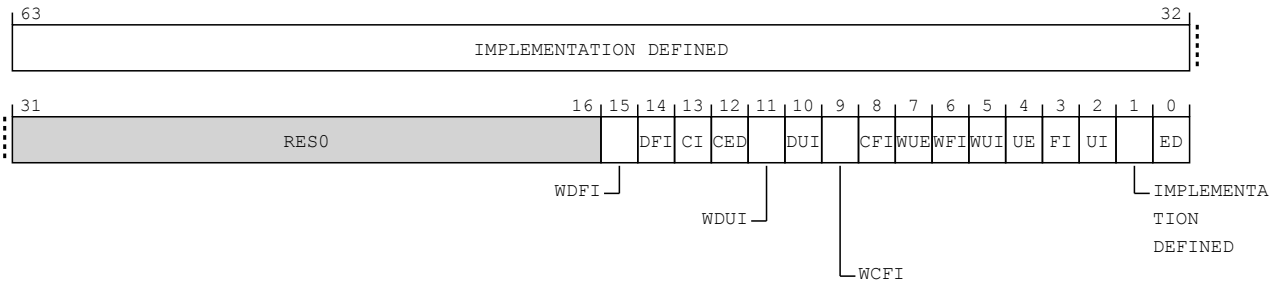
ERR<n>FR contains additional information about the node.

This register is present only when FEAT_RAS is implemented, error record n is implemented, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>CTLR are RES0.

Attributes

ERR<n>CTLR is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

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

Bits [31:16]

Reserved, RES0.

WDFI, bit [15]

When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == '11':

Fault handling interrupt for Deferred errors on writes enable, with ERR<n>CTLR.WFI.

When enabled by ERR<n>CTLR.{WDFI, WFI}:

- The fault handling interrupt is generated for errors recorded as Deferred error on writes.
- If the corresponding fault handling interrupt control for corrected error events, ERR<n>CTLR.WCFI, is not implemented, then the fault handling interrupt is generated for corrected error events on writes.

WDFI	Meaning
0b0	When ERR<n>CTLR.WFI == 0, Fault handling interrupt not generated for Deferred errors on writes. When ERR<n>CTLR.WFI == 1, Fault handling interrupt generated for Deferred errors on writes.
0b1	When ERR<n>CTLR.WFI == 0, Fault handling interrupt generated for Deferred errors on writes. When ERR<n>CTLR.WFI == 1, Fault handling interrupt not generated for Deferred errors on writes.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [14]

When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == '10':

DFI, bit [14]

Fault handling interrupt for Deferred errors enable, with ERR<n>CTLR.FI.

When ERR<n>FR.DFI == 0b10, this control applies to errors on both reads and writes.

When enabled by ERR<n>CTLR.{DFI, FI}:

- The fault handling interrupt is generated for all errors recorded as Deferred error.
- If the fault handling interrupt control for corrected error events, ERR<n>CTLR.CFI, is not implemented, then the fault handling interrupt is generated for all corrected error events.

DFI	Meaning
0b0	When ERR<n>CTLR.FI == 0, Fault handling interrupt not generated for Deferred errors. When ERR<n>CTLR.FI == 1, Fault handling interrupt generated for Deferred errors.
0b1	When ERR<n>CTLR.FI == 0, Fault handling interrupt generated for Deferred errors. When ERR<n>CTLR.FI == 1, Fault handling interrupt not generated for Deferred errors.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == '11':

RDFI, bit [14]

Fault handling interrupt for Deferred errors on reads enable, with ERR<n>CTLR.RFI.

When enabled by ERR<n>CTLR.{RDFI, RFI}:

- The fault handling interrupt is generated for errors recorded as Deferred error on reads.
- If the corresponding fault handling interrupt control for corrected error events, ERR<n>CTLR.RCFI, is not implemented, then the fault handling interrupt is generated for corrected error events on reads.

RDFI	Meaning
0b0	When ERR<n>CTLR.RFI == 0, Fault handling interrupt not generated for Deferred errors on reads. When ERR<n>CTLR.RFI == 1, Fault handling interrupt generated for Deferred errors on reads.
0b1	When ERR<n>CTLR.RFI == 0, Fault handling interrupt generated for Deferred errors on reads. When ERR<n>CTLR.RFI == 1, Fault handling interrupt not generated for Deferred errors on reads.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CI, bit [13]

When ERR<n>FR.CI == '10':

Critical error interrupt enable. When enabled, the critical error interrupt is generated for a critical error condition.

CI	Meaning
0b0	Critical error interrupt not generated for critical errors. Critical errors are treated as Uncontained errors.
0b1	Critical error interrupt generated for critical errors.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CED, bit [12]

When RAS System Architecture v2 is implemented, ERR<n>FR.CEC != '000', and ERR<n>FR.CED == '1':

Disable generation of corrected error events from error counters.

CED	Meaning
0b0	Corrected error events are generated by the error counter or counters.
0b1	Corrected error events are generated when a Corrected error is recorded.

See ERR<n>CTLR.CFI for more information on corrected error events.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WDUI, bit [11]

When ERR<n>FR.DUI == '11':

Error recovery interrupt for Deferred errors on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on writes.

WDUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors on writes.
0b1	Error recovery interrupt generated for Deferred errors on writes.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [10]

When ERR<n>FR.DUI == '10':

DUI, bit [10]

Error recovery interrupt for Deferred errors enable.

When ERR<n>FR.DUI == 0b10, this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Deferred error.

DUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors.
0b1	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.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.DUI == '11':

RDUI, bit [10]

Error recovery interrupt for Deferred errors on reads enable.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on reads.

RDUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors on reads.
0b1	Error recovery interrupt generated for Deferred errors on reads.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WCFI, bit [9]

When ERR<n>FR.CFI == '11':

Fault handling interrupt for corrected error events on writes enable.

When enabled, the fault handling interrupt is generated for corrected error events on writes.

WCFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events on writes.
0b1	Fault handling interrupt generated for corrected error events on writes.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

When ERR<n>FR.CFI == '10':

CFI, bit [8]

Fault handling interrupt for corrected error events enable.

When ERR<n>FR.CFI == 0b10, this control applies to errors on both reads and writes.

When enabled, the fault handling interrupt is generated for all corrected error events.

CFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events.
0b1	Fault handling interrupt generated for corrected error events.

If the node implements a corrected error counter or counters, and either ERR<n>CTLR.CED is not implemented or ERR<n>CTLR.CED is 0, then a corrected error event is defined as follows:

- A corrected error event occurs when a counter overflows and sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a software write sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a counter overflows and the overflow flag was previously set to 1.

Otherwise, a corrected error event occurs when the error record records an error as a Corrected error.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.CFI == '11':

RCFI, bit [8]

Fault handling interrupt for corrected error events on reads enable.

When enabled, the fault handling interrupt is generated for corrected error events on reads.

RCFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events on reads.
0b1	Fault handling interrupt generated for corrected error events on reads.

If the node implements a corrected error counter or counters, and either ERR<n>CTLR.CED is not implemented or ERR<n>CTLR.CED is 0, then a corrected error event is defined as follows:

- A corrected error event occurs when a counter overflows and sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a software write sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a counter overflows and the overflow flag was previously set to 1.

Otherwise, a corrected error event occurs when the error record records an error as a Corrected error.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUE, bit [7]

When ERR<n>FR.UE == '11':

In-band error response on writes enable.

When enabled, responses to writes that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

WUE	Meaning
0b0	In-band error response for uncorrected errors on writes disabled.
0b1	In-band error response for uncorrected errors on writes enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WFI, bit [6]

When ERR<n>FR.FI == '11':

Fault handling interrupt on writes enable.

When enabled:

- The fault handling interrupt is generated for errors recorded as Uncorrected error on writes.
- If the corresponding fault handling interrupt control for Deferred errors, ERR<n>CTLR.WDFI, is not implemented, then the fault handling interrupt is generated for errors recorded as Deferred error on writes.
- If the corresponding fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{WDFI, WCFI}, are not implemented, then the fault handling interrupt is generated for corrected error events on writes.

WFI	Meaning
0b0	Fault handling interrupt on writes disabled.
0b1	Fault handling interrupt on writes enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUI, bit [5]

When ERR<n>FR.UI == '11':

Uncorrected error recovery interrupt on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on writes.

WUI	Meaning
0b0	Error recovery interrupt on writes disabled.
0b1	Error recovery interrupt on writes enabled.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [4]

When ERR<n>FR.UE == '10':

UE, bit [4]

In-band error response enable.

When ERR<n>FR.UE == 0b10, this control applies to errors arising from both reads and writes.

When enabled, responses to transactions that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

UE	Meaning
0b0	In-band error response for uncorrected errors disabled.
0b1	In-band error response for uncorrected errors enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.UE == '11':

RUE, bit [4]

In-band error response on reads enable.

When enabled, responses to reads that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

RUE	Meaning
0b0	In-band error response for uncorrected errors on reads disabled.
0b1	In-band error response for uncorrected errors on reads enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [3]

When ERR<n>FR.FI == '10':

FI, bit [3]

Fault handling interrupt enable.

When ERR<n>FR.FI == 0b10, this control applies to errors on both reads and writes.

When enabled:

- The fault handling interrupt is generated for all errors recorded as Uncorrected error.
- If the fault handling interrupt control for Deferred errors, ERR<n>CTLR.DFI, is not implemented, then the fault handling interrupt is generated for all errors recorded as Deferred error.
- If the fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{DFI, CFI}, are not implemented, then the fault handling interrupt is generated for all corrected error events.

FI	Meaning
0b0	Fault handling interrupt disabled.
0b1	Fault handling interrupt enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.FI == '11':

RFI, bit [3]

Fault handling interrupt on reads enable.

When enabled:

- The fault handling interrupt is generated for errors recorded as Uncorrected error on reads.
- If the corresponding fault handling interrupt control for Deferred errors, ERR<n>CTLR.RDFI, is not implemented, then the fault handling interrupt is generated for errors recorded as Deferred error on reads.
- If the corresponding fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{RDFI, RCFI}, are not implemented, then the fault handling interrupt is generated for corrected error events on reads.

RFI	Meaning
0b0	Fault handling interrupt on reads disabled.
0b1	Fault handling interrupt on reads enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

When ERR<n>FR.UI == '10':

UI, bit [2]

Uncorrected error recovery interrupt enable.

When ERR<n>FR.UI == 0b10, this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Uncorrected error.

UI	Meaning
0b0	Error recovery interrupt disabled.
0b1	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.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.UI == '11':

RUI, bit [2]

Uncorrected error recovery interrupt on reads enable.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on reads.

RUI	Meaning
0b0	Error recovery interrupt on reads disabled.
0b1	Error recovery interrupt on reads enabled.

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

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMPLEMENTATION DEFINED, bit [1]

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

ED, bit [0]

When ERR<n>FR.ED == '10':

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, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection.

ED	Meaning
0b0	Error reporting disabled.
0b1	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. Uncorrected errors might result in corrupt data being silently propagated by the node.

Note

If this node requires initialization after Cold reset to prevent signaling false errors, then Arm recommends this field 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. If the Cold reset value is 1, the reset values of other controls in this register are also IMPLEMENTATION DEFINED and should not be UNKNOWN.

The reset behavior of this field is:

- On a Error recovery reset:
  - When RAS System Architecture v2 is implemented and ERR<n>FR.SRV == ‘1’, this field resets to '0'.

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERR<n>FR.SRV == ‘1’, this field resets to `'0'`.
  - Otherwise, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>CTLR

ERR<n>CTLR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x008 + (64 * n)`	ERR<n>CTLR

Accesses to this register are RW.

3.2.25 ERR<n>FR, Error Record <n> Feature Register, n = 0 - 65534

The ERR<n>FR characteristics are:

Purpose

Defines whether error record <n> is the first record owned by a node:

- If error record <n> is the first error record owned by a node, then ERR<n>FR.ED is not `0b00`.
- If error record <n> is not the first error record owned by a node, then ERR<n>FR.ED is `0b00`.

If error record <n> is the first record owned by the node, defines which of the common architecturally-defined features are implemented by the node and, of the implemented features, which are software programmable.

Configuration

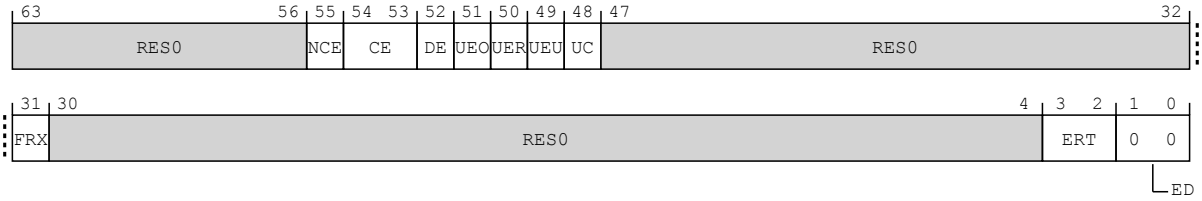
This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERR<n>FR are RES0.

Attributes

ERR<n>FR is a 64-bit register.

Field descriptions

When error record n is not implemented or error record n is not the first error record in the node:



Bits [63:56]

Reserved, RES0.

NCE, bit [55]

When ERR<n>FR.FRX == '1' and ERRFR[FirstRecordOfNode(n)].CEC != '000':

No countable errors. Describes whether this error record supports recording countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCE	Meaning
`0b0`	Records countable errors.
`0b1`	Does not record countable errors.

When ERRFR[FirstRecordOfNode(n)].CEC != `0b000`, at least one error record owned by the node records countable errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

CE, bits [54:53]

When ERR<n>FR.FRX == '1':

Corrected Error recording. Describes the types of Corrected errors the error record can record, if any.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	Does not record Corrected errors.
0b01	Records only transient or persistent Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to either 0b01 or 0b11.
0b10	Records only non-specific Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to 0b10.
0b11	Records all types of Corrected error.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DE, bit [52]

When ERR<n>FR.FRX == '1':

Deferred Error recording. Describes whether the error record supports recording Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	Does not record Deferred errors.
0b1	Records Deferred errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

UEO, bit [51]

When ERR<n>FR.FRX == '1':

Latent or Restartable Error recording. Describes whether the error record supports recording Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	Does not record Latent or Restartable errors.
0b1	Records Latent or Restartable errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

UER, bit [50]

When ERR<n>FR.FRX == '1':

Signaled or Recoverable Error recording. Describes whether the error record supports recording Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	Does not record Signaled or Recoverable errors.
0b1	Records Signaled or Recoverable errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

UEU, bit [49]

When ERR<n>FR.FRX == '1':

Unrecoverable Error recording. Describes whether the error record supports recording Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	Does not record Unrecoverable errors.
0b1	Records Unrecoverable errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

UC, bit [48]

When ERR<n>FR.FRX == '1':

Uncontainable Error recording. Describes whether the error record supports recording Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	Does not record Uncontainable errors.
0b1	Records Uncontainable errors.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [47:32]

Reserved, RES0.

FRX, bit [31]

When error record n is implemented, RAS System Architecture v2 is implemented, and ERR<n>FR.ERT == '00':

Feature Register extension. Defines whether ERR<n>FR[63:48] describe architecturally-defined properties of this error record, including the supported error types.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRX	Meaning
0b0	ERR<n>FR[63:48] are RES0.
0b1	ERR<n>FR[63:48] are defined by the architecture.

If ERR<n>FR.FRX is 0, error record <n> is implemented, and ERRFR[FirstRecordOfNode(n)].FRX is 1, then ERRFR[FirstRecordOfNode(n)][63:48] describe the architecturally-defined properties of all error records owned by the node.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [30:4]

Reserved, RES0.

ERT, bits [3:2]

When RAS System Architecture v2 is implemented:

Error Record Type. Defines the type of error record.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ERT	Meaning
0b00	Error record <n> not implemented or is a normal record that is not the first error record of the node.
0b01	Error record <n> is a continuation record of the previous error record, <n-1>.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

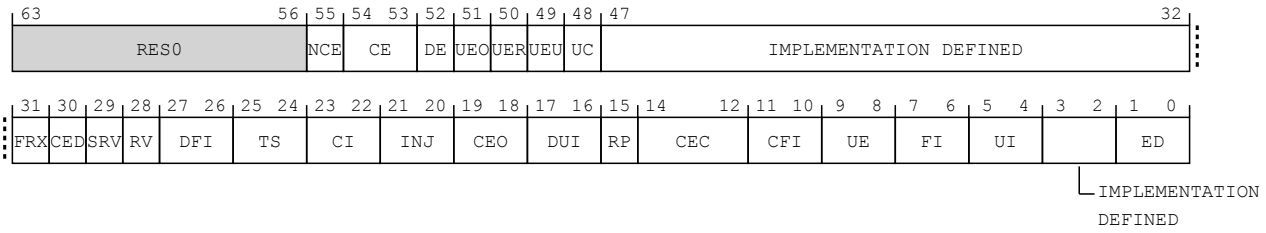
ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is not the first error record owned the node.

ED	Meaning
0b00	Error record <n> is not implemented or is not the first error record owned by the node.

Access to this field is RO.

When error record n is the first error record in the node:



Bits [63:56]

When ERR<n>FR.FRX == '1':

Bits [63:56]

Reserved, RES0.

Otherwise:

Bits [63:56]

Reserved for identifying IMPLEMENTATION DEFINED controls.

NCE, bit [55]

When RAS System Architecture v2 is implemented, ERR<n>FR.FRX == '1', and ERR<n>FR.CEC != '000':

No countable errors. Describes whether this error record supports recording countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCE	Meaning
0b0	Records countable errors.
0b1	Does not record countable errors.

When ERR<n>FR.CEC != 0b000, at least one error record owned by the node records countable errors.

Access to this field is RO.

When ERR<n>FR.FRX == '1':

Reserved, RES0.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

CE, bits [54:53]

When ERR<n>FR.FRX == '1':

Corrected Error recording. Describes the types of Corrected errors the node can record, if any.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	Does not record Corrected errors.
0b01	Records only transient or persistent Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to either 0b01 or 0b11.
0b10	Records only non-specific Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to 0b10.
0b11	Records all types of Corrected error.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

DE, bit [52]

When ERR<n>FR.FRX == '1':

Deferred Error recording. Describes whether the node supports recording Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	Does not record Deferred errors.
0b1	Records Deferred errors.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEO, bit [51]

When ERR<n>FR.FRX == '1':

Latent or Restartable Error recording. Describes whether the node supports recording Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	Does not record Latent or Restartable errors.
0b1	Records Latent or Restartable errors.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UER, bit [50]

When ERR<n>FR.FRX == '1':

Signaled or Recoverable Error recording. Describes whether the node supports recording Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	Does not record Signaled or Recoverable errors.
0b1	Records Signaled or Recoverable errors.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEU, bit [49]

When ERR<n>FR.FRX == '1':

Unrecoverable Error recording. Describes whether the node supports recording Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	Does not record Unrecoverable errors.
0b1	Records Unrecoverable errors.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UC, bit [48]

When ERR<n>FR.FRX == '1':

Uncontainable Error recording. Describes whether the node supports recording Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	Does not record Uncontainable errors.
0b1	Records Uncontainable errors.

Access to this field is RO.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

IMPLEMENTATION DEFINED, bits [47:32]

Reserved for identifying IMPLEMENTATION DEFINED controls.

FRX, bit [31]

When RAS System Architecture v1p1 is implemented:

Feature Register extension.

Defines whether ERR<n>FR[63:48] describe architecturally-defined properties of this error record or node, including the supported error types, or describe IMPLEMENTATION DEFINED properties.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRX	Meaning
0b0	ERR<n>FR[63:48] are IMPLEMENTATION DEFINED.
0b1	ERR<n>FR[63:48] are defined by the architecture.

When ERR<n>FR.FRX is 1:

- If RAS System Architecture v2 is implemented and ERR<m>FR.FRX is 1 for other error records <m> owned by the same node, then ERR<n>FR[63:48] describe the architecturally-defined properties of error record <n> only, and ERR<m>FR[63:48] describe the properties for error record <m>.

- Otherwise, ERR<n>FR[63:48] describe the architecturally-defined properties of all error records owned by the node.

Access to this field is RO.

Otherwise:

Reserved, RES0.

CED, bit [30]

When RAS System Architecture v2 is implemented and ERR<n>FR.CEC != '000':

Error counter disable. Indicates whether the node implements a control to disable any implemented Corrected error counters.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CED	Meaning
0b0	Error counter disable control is not implemented and the error counter(s) are always enabled. ERR<n>CTLR.CED is RES0.
0b1	Enabling and disabling of error counter(s) is supported and controlled by ERR<n>CTLR.CED.

Access to this field is RO.

Otherwise:

Reserved, RES0.

SRV, bit [29]

When RAS System Architecture v2 is implemented:

Status Reset Value. Indicates how node <n> and each error record <m> owned by node <n> is reset.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SRV	Meaning
0b0	Node <n> and each error record <m> owned by node <n> are reset as follows: • ERR<m>STATUS.{AV, V, MV} are set to {0, 0, 0} on a Cold reset and preserved on Error Recovery reset. • ERR<n>CTLR.ED is set to an IMPLEMENTATION DEFINED value on a Cold reset and preserved on Error Recovery reset.
0b1	Node <n> and each error record <m> owned by node <n> are reset as follows: • ERR<m>STATUS.{AV, V, MV} are set to architecturally UNKNOWN values on a Cold reset and preserved on Error Recovery reset. • ERR<n>CTLR.ED is set to 0 on both Cold reset and Error Recovery reset.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

RV, bit [28]

When RAS System Architecture v2 is implemented:

Reset Valid. Indicates whether each error record <m> implemented by the node includes the Reset Valid flags, ERR<m>STATUS.{RV, RV2}.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RV	Meaning
0b0	ERR<m>STATUS.{RV, RV2} are RES0.
0b1	ERR<m>STATUS.{RV, RV2} are R/W1C bits. See ERR<m>STATUS.{RV, RV2} for more information.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DFI, bits [27:26]

When RAS System Architecture v2 is implemented and !(ERR<n>FR.FI IN {'0x'}):

Fault handling interrupt for deferred errors control. Indicates whether the enabling and disabling of fault handling interrupts on deferred errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DFI	Meaning
0b00	Does not support the enabling and disabling of fault handling interrupts on deferred errors. ERR<n>CTLR.DFI is RES0.
0b10	Enabling and disabling of fault handling interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DFI.
0b11	Enabling and disabling of fault handling interrupts on deferred errors is supported, and controllable using ERR<n>CTLR.WDFI for writes and ERR<n>CTLR.RDFI for reads.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

TS, bits [25:24]

Timestamp Extension. Indicates whether, for each error record <m> owned by this node, ERR<m>MISC3 is used as the timestamp register, and, if it is, the timebase used by the timestamp.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TS	Meaning
0b00	Does not support a timestamp register.
0b01	Implements a timestamp register in ERR<n>MISC3 for each error record <m> owned by the node. The timestamp uses the same timebase as the system Generic Timer. **Note** For an error record that has an affinity to a PE, this is the same timer that is visible through CNTPCT_EL0 at the highest Exception level on that PE.
0b10	Implements a timestamp register in ERR<m>MISC3 for each error record <m> owned by the node. The timestamp uses an IMPLEMENTATION DEFINED timebase.

All other values are reserved.

Access to this field is RO.

CI, bits [23:22]

Critical error interrupt. Indicates whether the critical error interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CI	Meaning
0b00	Does not support the critical error interrupt. ERR<n>CTLR.CI is RES0.
0b01	Critical error interrupt is supported and always enabled. ERR<n>CTLR.CI is RES0.
0b10	Critical error interrupt is supported and controllable using ERR<n>CTLR.CI.

All other values are reserved.

Access to this field is RO.

INJ, bits [21:20]

Fault Injection Extension. Indicates whether the Common Fault Injection Model Extension is implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INJ	Meaning
0b00	Does not support the Common Fault Injection Model Extension.
0b01	Supports the Common Fault Injection Model Extension. See ERR<n>PFGF for more information.

All other values are reserved.

Access to this field is RO.

CEO, bits [19:18]

When ERR<n>FR.CEC != '000':

Corrected Error overwrite. Indicates the behavior of the node when a second or subsequent Corrected error is recorded and a first Corrected error has previously been recorded by an error record <m> owned by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CEO	Meaning
0b00	Keeps the previous error syndrome.
0b01	If ERR<m>STATUS.OF is 1 before the Corrected error is counted, then the error record keeps the previous syndrome. Otherwise the previous syndrome is overwritten.

All other values are reserved.

The second or subsequent Corrected error is counted by the Corrected error counter, regardless of the value of this field. If counting the error causes unsigned overflow of the counter, then ERR<m>STATUS.OF is set to 1.

This means that, if no other error is subsequently recorded that overwrites the syndrome:

- If ERR<n>FR.CEO is 0b00, the error record holds the syndrome for the first recorded Corrected error.
- If ERR<n>FR.CEO is 0b01, the error record holds the syndrome for the most recently recorded Corrected error before the counter overflows.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DUI, bits [17:16]

When ERR<n>FR.UI != '00':

Error recovery interrupt for deferred errors control. Indicates whether the enabling and disabling of error recovery interrupts on deferred errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DUI	Meaning
0b00	Does not support the enabling and disabling of error recovery interrupts on deferred errors. ERR<n>CTLR.DUI is RES0.
0b10	Enabling and disabling of error recovery interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DUI.
0b11	Enabling and disabling of error recovery interrupts on deferred errors is supported, and controllable using ERR<n>CTLR.WDUI for writes and ERR<n>CTLR.RDUI for reads.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

RP, bit [15]

When ERR<n>FR.CEC != '000':

Repeat counter. Indicates whether the node implements a second Corrected error counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RP	Meaning
0b0	Implements a single Corrected error counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.
0b1	Implements a first (repeat) counter and a second (other) counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors. The repeat counter is the same size as the primary error counter.

Access to this field is RO.

Otherwise:

Reserved, RES0.

CEC, bits [14:12]

Corrected Error Counter. Indicates whether the node implements the standard format Corrected error counter mechanisms in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CEC	Meaning
0b000	Does not implement the standard format Corrected error counter model.
0b010	Implements an 8-bit Corrected error counter in ERR<m>MISC0[39:32] for each error record <m> owned by the node that can record countable errors.
0b100	Implements a 16-bit Corrected error counter in ERR<m>MISC0[47:32] for each error record <m> owned by the node that can record countable errors.

All other values are reserved.

Note

Implementations might include other error counter models, or might include the standard format model and not indicate this in ERR<n>FR.

Access to this field is RO.

CFI, bits [11:10]

When !(ERR<n>FR.FI IN {'0x'}):

Fault handling interrupt for corrected errors control. Indicates whether the enabling and disabling of fault handling interrupts on corrected errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CFI	Meaning
0b00	Does not support the enabling and disabling of fault handling interrupts on corrected errors. ERR<n>CTLR.CFI is RES0.
0b10	Enabling and disabling of fault handling interrupts on corrected errors is supported and controllable using ERR<n>CTLR.CFI.
0b11	Enabling and disabling of fault handling interrupts on corrected errors is supported, and controllable using ERR<n>CTLR.WCFI for writes and ERR<n>CTLR.RCFI for reads.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

UE, bits [9:8]

In-band error response (External abort). Indicates whether the in-band error response and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UE	Meaning
0b00	Does not support the in-band error response. ERR<n>CTLR.UE is RES0.
0b01	In-band error response is supported and always enabled. ERR<n>CTLR.UE is RES0.
0b10	In-band error response is supported and controllable using ERR<n>CTLR.UE.
0b11	In-band error response is supported, and controllable using ERR<n>CTLR.WUE for writes and ERR<n>CTLR.RUE for reads.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

Access to this field is RO.

FI, bits [7:6]

Fault handling interrupt. Indicates whether the fault handling interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FI	Meaning
0b00	Does not support the fault handling interrupt. ERR<n>CTLR.FI is RES0.
0b01	Fault handling interrupt is supported and always enabled. ERR<n>CTLR.FI is RES0.
0b10	Fault handling interrupt is supported and controllable using ERR<n>CTLR.FI.
0b11	Fault handling interrupt is supported, and controllable using ERR<n>CTLR.WFI for writes and ERR<n>CTLR.RFI for reads.

Access to this field is RO.

UI, bits [5:4]

Error recovery interrupt for uncorrected errors. Indicates whether the error handling interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UI	Meaning
0b00	Does not support the error handling interrupt. ERR<n>CTLR.UI is RES0.
0b01	Error handling interrupt is supported and always enabled. ERR<n>CTLR.UI is RES0.
0b10	Error handling interrupt is supported and controllable using ERR<n>CTLR.UI.
0b11	Error handling interrupt is supported, and controllable using ERR<n>CTLR.WUI for writes and ERR<n>CTLR.RUI for reads.

Access to this field is RO.

IMPLEMENTATION DEFINED, bits [3:2]

IMPLEMENTATION DEFINED.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is a normal record and the first record owned the node, and whether the node implements the controls for enabling and disabling error reporting and logging.

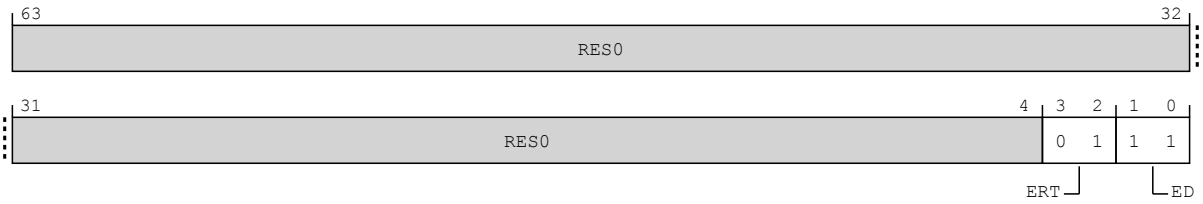
The value of this field is an IMPLEMENTATION DEFINED choice of:

ED	Meaning
0b01	Error reporting and logging always enabled. ERR<n>CTLR.ED is RES0.
0b10	Error reporting and logging is controllable using ERR<n>CTLR.ED.

All other values are reserved.

Access to this field is RO.

When RAS System Architecture v2 is implemented and error record <n> is a proxy for a RAS agent:



Bits [63:4]

Reserved, RES0.

ERT, bits [3:2]

Error Record Type. Defines the type of error record.

ERT	Meaning
0b01	Error record is a proxy for a RAS agent.

All other values are reserved.

Access to this field is RO.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is not a true error record.

ED	Meaning
0b11	Error record <n> is not an error record.

Access to this field is RO.

Accessing ERR<n>FR

ERR<n>FR can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0x000 + (64 * n)	ERR<n>FR

Accesses to this register are RO.

3.2.26 ERR<n>MISC0, Error Record <n> Miscellaneous Register 0, n = 0 - 65534

The ERR<n>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record <n> implements a standard format Corrected error counter or counters (ERRFR[FirstRecordOfNode(n)].CEC != `0b000`), then it is IMPLEMENTATION DEFINED whether error record <n> can record countable errors, and:

- If error record <n> records countable errors, then ERR<n>MISC0 implements the standard format Corrected error counter or counters for error record <n>.
- If error record <n> does not record countable errors, then it is recommended that the fields in ERR<n>MISC0 defined for the standard format counter or counters are RES0. That is, the fields behave like counters that never count.

Configuration

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC0, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

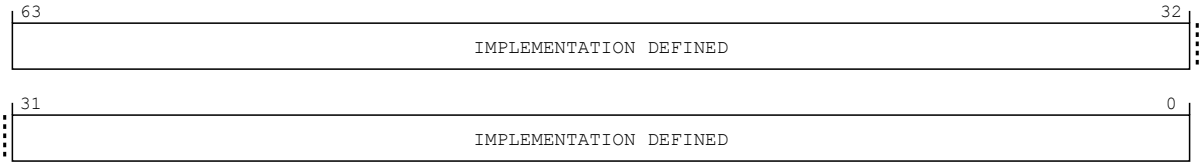
This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>MISC0 are RES0.

Attributes

ERR<n>MISC0 is a 64-bit register.

Field descriptions

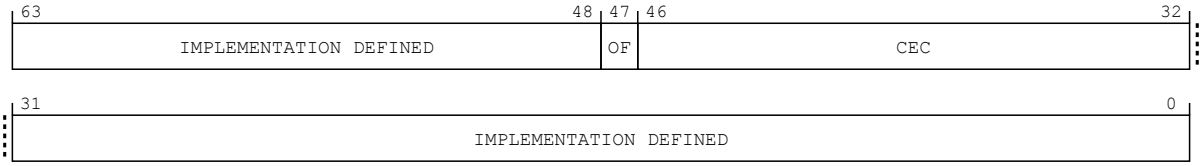
When ERRFR[FirstRecordOfNode(n)].CEC == '000' or error record n does not record countable errors:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == '100', ERRFR[FirstRecordOfNode(n)].RP == '0', and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OF, bit [47]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

OF	Meaning
0b0	Counter has not overflowed.
0b1	Counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [46:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

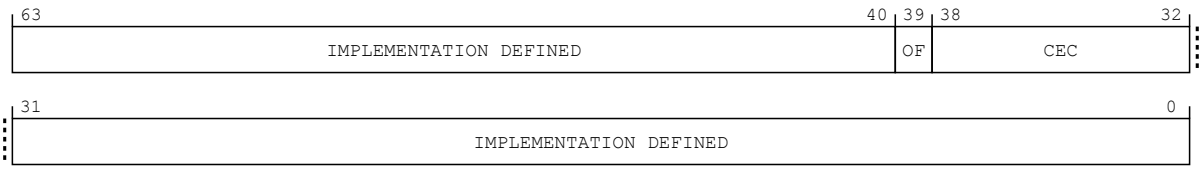
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == '010', ERRFR[FirstRecordOfNode(n)].RP == '0', and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:40]

IMPLEMENTATION DEFINED syndrome.

OF, bit [39]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

OF	Meaning
0b0	Counter has not overflowed.
0b1	Counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [38:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

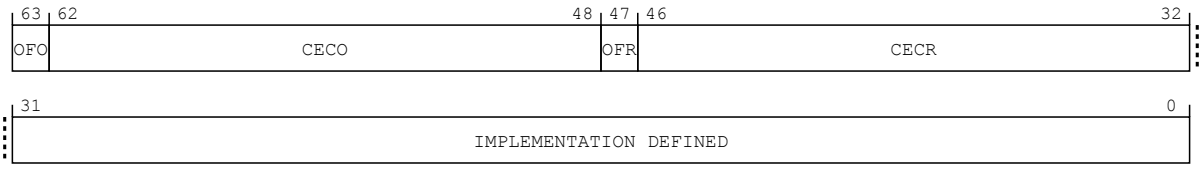
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == '100', ERRFR[FirstRecordOfNode(n)].RP == '1', and error record n records countable errors:



OFO, bit [63]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

OFO	Meaning
0b0	Other counter has not overflowed.
0b1	Other counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [62:48]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [47]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

OFR	Meaning
0b0	Repeat counter has not overflowed.
0b1	Repeat counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [46:32]

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.

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 ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

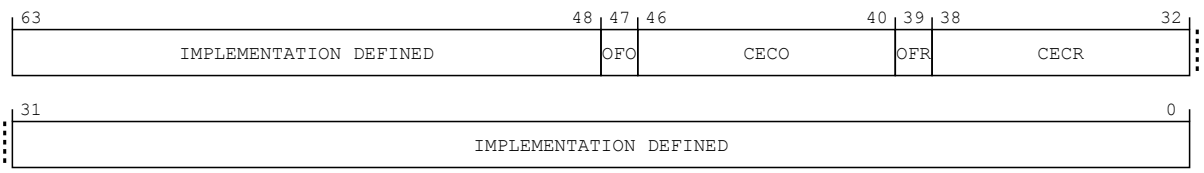
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == '010', ERRFR[FirstRecordOfNode(n)].RP == '1', and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OFO, bit [47]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

OFO	Meaning
0b0	Other counter has not overflowed.
0b1	Other counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [46:40]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [39]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

OFR	Meaning
0b0	Repeat counter has not overflowed.
0b1	Repeat counter has overflowed.

A direct write that modifies this field 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 field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [38:32]

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.

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 ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC0

Reads from ERR<n>MISC0 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x020 + (64 * n)`	ERR<n>MISC0

Accesses to this register are RW.

3.2.27 ERR<n>MISC1, Error Record <n> Miscellaneous Register 1, n = 0 - 65534

The ERR<n>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC1, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Note

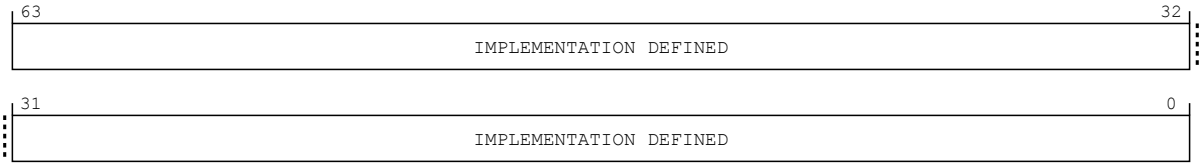
Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>MISC1 are RES0.

Attributes

ERR<n>MISC1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC1

Reads from ERR<n>MISC1 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x028 + (64 * n)`	ERR<n>MISC1

Accesses to this register are RW.

3.2.28 ERR<n>MISC2, Error Record <n> Miscellaneous Register 2, n = 0 - 65534

The ERR<n>MISC2 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC2, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Arm recommends that if RAS System Architecture v1.1 is not implemented, then ERR<n>MISC2 does not require zeroing to return the record to a quiescent state.

Note

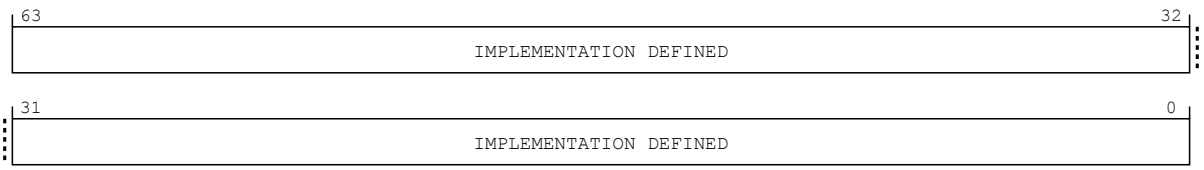
Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

This register is present only when IsFeatureImplemented(FEAT_RAS) && ((ImpDefBool("IMPLEMENTED_ERR<n>MISC2") || IsFeatureImplemented(FEAT_RASSAv1p1)) && IsErrorRecordImplemented(n)). Otherwise, direct accesses to ERR<n>MISC2 are RES0.

Attributes

ERR<n>MISC2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC2

Reads from ERR<n>MISC2 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x030 + (64 * n)`	ERR<n>MISC2

Accesses to this register are RW.

3.2.29 ERR<n>MISC3, Error Record <n> Miscellaneous Register 3, n = 0 - 65534

The ERR<n>MISC3 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record n supports the RAS Timestamp Extension (ERRFR[FirstRecordOfNode(n)].TS != `0b00`), then ERR<n>MISC3 contains the timestamp value for error record n when the error was detected. Otherwise the contents of ERR<n>MISC3 are IMPLEMENTATION DEFINED.

Configuration

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC3, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Arm recommends that if RAS System Architecture v1.1 is not implemented, then ERR<n>MISC3 does not require zeroing to return the record to a quiescent state.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

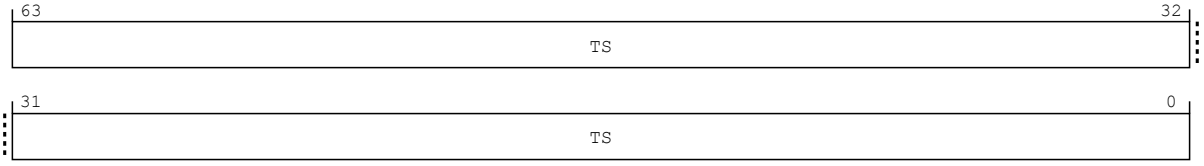
This register is present only when IsFeatureImplemented(FEAT_RAS) && ((ImpDefBool("IMPLEMENTED_ERR<n>MISC3") || IsFeatureImplemented(FEAT_RASSAv1p1)) && IsErrorRecordImplemented(n)). Otherwise, direct accesses to ERR<n>MISC3 are RES0.

Attributes

ERR<n>MISC3 is a 64-bit register.

Field descriptions

When ERRFR[FirstRecordOfNode(n)].TS != '00':



TS, bits [63:0]

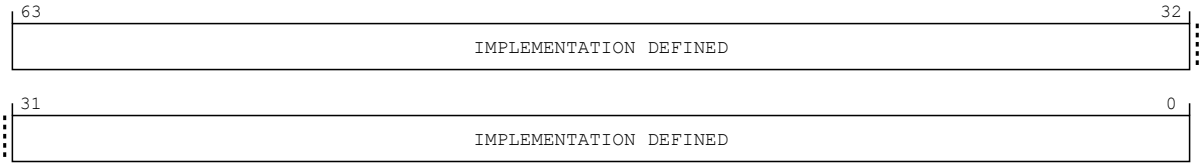
Timestamp. Timestamp value recorded when the error was detected. Valid only if ERR<n>STATUS.V == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO or RW.

When ERRFR[FirstRecordOfNode(n)].TS == '00':



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC3

Reads from ERR<n>MISC3 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC3 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x038 + (64 * n)`	ERR<n>MISC3

Accesses to this register are RW.

3.2.30 ERR<n>PFGCDN, Error Record <n> Pseudo-fault Generation Countdown Register, n = 0 - 65534

The ERR<n>PFGCDN characteristics are:

Purpose

Generates one of the errors enabled in the corresponding ERR<n>PFGCTL register.

Configuration

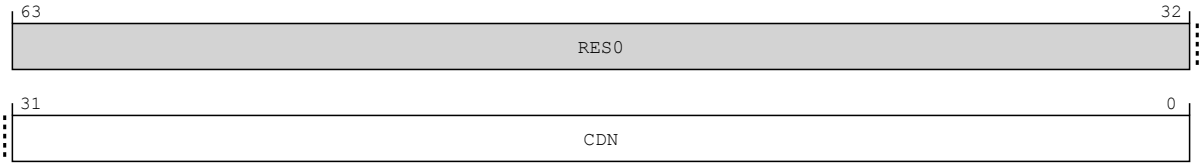
ERR<n>FR describes the features implemented by the node.

This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGCDN are RES0.

Attributes

ERR<n>PFGCDN is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

CDN, bits [31:0]

Countdown value.

This field is copied to Error Generation Counter when either:

- Software writes 1 to ERR<n>PFGCTL.CDNEN.
- The Error Generation Counter decrements to zero and ERR<n>PFGCTL.R is 1.

While ERR<n>PFGCTL.CDNEN is 1 and the Error Generation Counter is nonzero, the counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate. When the counter reaches zero, one of the errors enabled in the ERR<n>PFGCTL register is generated.

Note

The current Error Generation Counter value is not visible to software.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>PFGCDN

This section shows the offset of ERR<n>PFGCDN in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGCDN is accessed in a fault injection group, see ‘RAS memory-mapped register views’ for the offset of ERR<n>PFGCDN.

ERR<n>PFGCDN can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x810 + (64 * n)`	ERR<n>PFGCDN

Accesses to this register are RW.

3.2.31 ERR<n>PFGCTL, Error Record <n> Pseudo-fault Generation Control Register, n = 0 - 65534

The ERR<n>PFGCTL characteristics are:

Purpose

Enables controlled fault generation.

Configuration

ERR<n>PFGF describes the Common Fault Injection features implemented by the node.

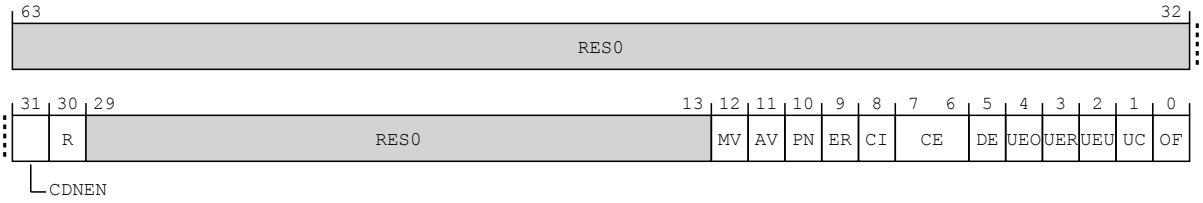
ERR<n>FR describes the features implemented by the node.

This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGCTL are RES0.

Attributes

ERR<n>PFGCTL is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

CDNEN, bit [31]

Countdown Enable. Controls transfers of the value held in ERR<n>PFGCDN to the Error Generation Counter and enables this counter.

CDNEN	Meaning
0b0	The Error Generation Counter is disabled.
0b1	The Error Generation Counter is enabled. On a write of 1 to this field, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

R, bit [30]

When ERR<n>PFGF.R == '1':

Restart. Controls whether the Error Generation Counter restarts or stops counting on reaching zero.

R	Meaning
0b0	On reaching zero, the Error Generation Counter will stop counting.
0b1	On reaching zero, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:13]

Reserved, RES0.

MV, bit [12]

When ERR<n>PFGF.MV == '1':

Miscellaneous syndrome. The value written to ERR<n>STATUS.MV when an injected error is recorded.

MV	Meaning
0b0	ERR<n>STATUS.MV is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.MV is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the node always sets ERRSTATUS.MV to 1 when an injected error is recorded, access to this field is RAO/WI.

When the node always sets ERR<n>STATUS.MV to 1 when an injected error is recorded and this field is RAO/WI:

Reserved, RAO/WI.

Otherwise:

Reserved, RES0.

AV, bit [11]

When ERR<n>PFGF.AV == '1':

Address syndrome. The value written to ERR<n>STATUS.AV when an injected error is recorded.

AV	Meaning
0b0	ERR<n>STATUS.AV is set to 0 when an injected error is recorded.

AV	Meaning
0b1	ERR<n>STATUS.AV is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the node always sets ERRSTATUS.AV to 1 when an injected error is recorded, access to this field is RAO/WI.

When the node always sets ERR<n>STATUS.AV to 1 when an injected error is recorded and this field is RAO/WI:

Reserved, RAO/WI.

Otherwise:

Reserved, RES0.

PN, bit [10]

When ERR<n>PFGF.PN == '1':

Poison flag. The value written to ERR<n>STATUS.PN when an injected error is recorded.

PN	Meaning
0b0	ERR<n>STATUS.PN is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.PN is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ER, bit [9]

When ERR<n>PFGF.ER == '1':

Error Reported flag. The value written to ERR<n>STATUS.ER when an injected error is recorded.

ER	Meaning
0b0	ERR<n>STATUS.ER is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.ER is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CI, bit [8]

When ERR<n>PFGF.CI == '1':

Critical Error flag. The value written to ERR<n>STATUS.CI when an injected error is recorded.

CI	Meaning
0b0	ERR<n>STATUS.CI is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.CI is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CE, bits [7:6]

When ERR<n>PFGF.CE != '00':

Corrected Error generation enable. Controls the type of injected Corrected error generated by the fault injection feature of the node.

CE	Meaning	Applies when
0b00	An injected Corrected error will not be generated by the fault injection feature of the node.	
0b01	An injected non-specific Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b10 when the injected error is recorded.	ERRPFGF.CE == '01'
0b10	An injected transient Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b01 when the injected error is recorded.	ERRPFGF.CE == '11'
0b11	An injected persistent Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b11 when the injected error is recorded.	ERRPFGF.CE == '11'

The set of permitted values for this field is defined by ERR<n>PFGF.CE.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DE, bit [5]

When ERR<n>PFGF.DE == '1':

Deferred Error generation enable. Controls whether an injected Deferred error is generated by the fault injection feature of the node.

DE	Meaning
0b0	An injected Deferred error will not be generated by the fault generation feature of the node.
0b1	An injected Deferred error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEO, bit [4]

When ERR<n>PFGF.UEO == '1':

Latent or Restartable Error generation enable. Controls whether an injected Latent or Restartable error is generated by the fault injection feature of the node.

UEO	Meaning
0b0	An injected Latent or Restartable error will not be generated by the fault generation feature of the node.
0b1	An injected Latent or Restartable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UER, bit [3]

When ERR<n>PFGF.UER == '1':

Signaled or Recoverable Error generation enable. Controls whether an injected Signaled or Recoverable error is generated by the fault injection feature of the node.

UER	Meaning
0b0	An injected Signaled or Recoverable error will not be generated by the fault generation feature of the node.
0b1	An injected Signaled or Recoverable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEU, bit [2]

When ERR<n>PFGF.UEU == '1':

Unrecoverable Error generation enable. Controls whether an injected Unrecoverable error is generated by the fault injection feature of the node.

UEU	Meaning
0b0	An injected Unrecoverable error will not be generated by the fault generation feature of the node.
0b1	An injected Unrecoverable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UC, bit [1]

When ERR<n>PFGF.UC == '1':

Uncontainable Error generation enable. Controls whether an injected Uncontainable error is generated by the fault injection feature of the node.

UC	Meaning
0b0	An injected Uncontainable error will not be generated by the fault generation feature of the node.
0b1	An injected Uncontainable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OF, bit [0]

When ERR<n>PFGF.OF == '1':

Overflow flag. The value written to ERR<n>STATUS.OF when an injected error is recorded.

OF	Meaning
0b0	ERR<n>STATUS.OF is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.OF is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>PFGCTL

This section shows the offset of ERR<n>PFGCTL in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGCTL is accessed in a fault injection group, see ‘RAS memory-mapped register views’ for the offset of ERR<n>PFGCTL.

ERR<n>PFGCTL can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0x808 + (64 * n)	ERR<n>PFGCTL

Accesses to this register are RW.

3.2.32 ERR<n>PFGF, Error Record <n> Pseudo-fault Generation Feature Register, n = 0 - 65534

The ERR<n>PFGF characteristics are:

Purpose

Defines which common architecturally-defined fault generation features are implemented.

Configuration

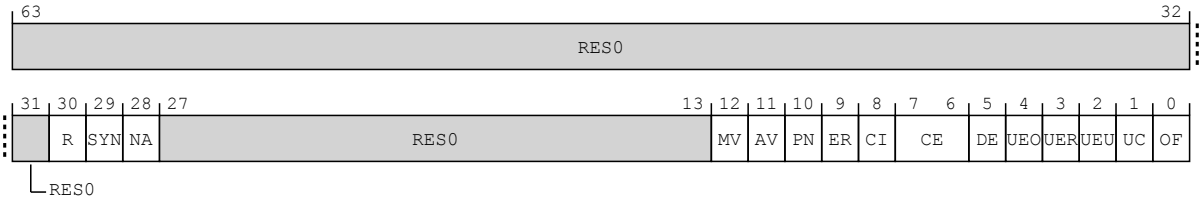
ERR<n>FR describes the features implemented by the node.

This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGF are RES0.

Attributes

ERR<n>PFGF is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

R, bit [30]

Restartable. Support for Error Generation Counter restart mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

R	Meaning
0b0	The node does not support this feature. ERR<n>PFGCTL.R is RES0.
0b1	Error Generation Counter restart mode is implemented and is controlled by ERR<n>PFGCTL.R. ERR<n>PFGCTL.R is a read/write field.

Access to this field is RO.

SYN, bit [29]

Syndrome. Fault syndrome injection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYN	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.{IERR, SERR} to IMPLEMENTATION DEFINED values. ERR<n>STATUS.{IERR, SERR} are UNKNOWN when ERR<n>STATUS.V is 0.
0b1	When an injected error is recorded, the node does not update the ERR<n>STATUS.{IERR, SERR} fields. ERR<n>STATUS.{IERR, SERR} are writable when ERR<n>STATUS.V is 0.

Note

If ERR<n>PFGF.SYN is 1 then software can write specific values into the ERR<n>STATUS.{IERR, SERR} fields when setting up a fault injection event. The sets of values that can be written to these fields is IMPLEMENTATION DEFINED.

Access to this field is RO.

NA, bit [28]

No access required. Defines whether this component fakes detection of the error on an access to the component or spontaneously in the fault injection state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NA	Meaning
0b0	The component fakes detection of the error on an access to the component.
0b1	The component fakes detection of the error spontaneously in the fault injection state.

Access to this field is RO.

Bits [27:13]

Reserved, RES0.

MV, bit [12]

Miscellaneous syndrome.

Defines whether software can control all or part of the syndrome recorded in the ERR<n>MISC<m> registers when an injected error is recorded.

It is IMPLEMENTATION DEFINED which ERR<n>MISC<m> syndrome fields, if any, are updated by the node when an injected error is recorded. Some syndrome fields might always be updated by the node when an error, including an injected error, is recorded. For example, a corrected error counter might always be updated when any countable error, including a injected countable error, is recorded.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MV	Meaning
0b0	When an injected error is recorded, the node might update the ERR<n>MISC<m> registers: • If any syndrome is recorded by the node in the ERR<n>MISC<m> registers, then ERR<n>STATUS.MV is set to 1. • Otherwise, ERR<n>STATUS.MV is unchanged. If the node always sets ERR<n>STATUS.MV to 1 when recording an injected error then ERR<n>PFGCTL.MV might be RAO/WI. Otherwise ERR<n>PFGCTL.MV is RES0.
0b1	When an injected error is recorded, the node might update some, but not all ERR<n>MISC<m> syndrome fields: • If any syndrome is recorded by the node in the ERR<n>MISC<m> registers, then ERR<n>STATUS.MV is set to 1. • Otherwise, ERR<n>STATUS.MV is set to ERR<n>PFGCTL.MV. ERR<n>MISC<m> syndrome fields that are not updated by the node are writable when ERR<n>STATUS.MV is 0. If the node always sets ERR<n>STATUS.MV to 1 when recording an injected error then ERR<n>PFGCTL.MV is RAO/WI. Otherwise ERR<n>PFGCTL.MV is a read/write field.

If ERR<n>PFGF.MV is 1, software can write specific additional syndrome values into the ERR<n>MISC<m> registers when setting up a fault injection event. The permitted values that can be written to these registers are IMPLEMENTATION DEFINED.

Access to this field is RO.

AV, bit [11]

Address syndrome. Defines whether software can control the address recorded in ERR<n>ADDR when an injected error is recorded.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AV	Meaning
0b0	When an injected error is recorded, the node might record an address in ERR<n>ADDR. If an address is recorded in ERR<n>ADDR, then ERR<n>STATUS.AV is set to 1. Otherwise, ERR<n>ADDR and ERR<n>STATUS.AV are unchanged. If the node always records an address and sets ERR<n>STATUS.AV to 1 when recording an injected error then ERR<n>PFGCTL.AV might be RAO/WI. Otherwise ERR<n>PFGCTL.AV is RES0.
0b1	When an injected error is recorded, the node does not update ERR<n>ADDR and does one of: • Sets ERR<n>STATUS.AV to ERR<n>PFGCTL.AV. ERR<n>PFGCTL.AV is a read/write field. • Sets ERR<n>STATUS.AV to 1. ERR<n>PFGCTL.AV is RAO/WI. ERR<n>ADDR is writable when ERR<n>STATUS.AV is 0.

If ERR<n>PFGF.AV is 1 then software can write a specific address value into ERR<n>ADDR when setting up a fault injection event.

Access to this field is RO.

PN, bit [10]

When the node supports this flag:

Poison flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.PN status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PN	Meaning
0b0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.PN to 1. ERR<n>PFGCTL.PN is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.PN is set to ERR<n>PFGCTL.PN. ERR<n>PFGCTL.PN is a read/write field.

This behavior replaces the architecture-defined rules for setting the ERR<n>STATUS.PN bit.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

ER, bit [9]

When the node supports this flag:

Error Reported flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.ER status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ER	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.ER according to the architecture-defined rules for setting the ER field. ERR<n>PFGCTL.ER is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.ER is set to ERR<n>PFGCTL.ER. This behavior replaces the architecture-defined rules for setting the ER bit. ERR<n>PFGCTL.ER is a read/write field.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

CI, bit [8]

When the node supports this flag:

Critical Error flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.CI status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CI	Meaning
0b0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.CI to 1. ERR<n>PFGCTL.CI is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.CI is set to ERR<n>PFGCTL.CI. ERR<n>PFGCTL.CI is a read/write field.

This behavior replaces the architecture-defined rules for setting the ERR<n>STATUS.CI bit.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

CE, bits [7:6]

When the node supports this type of error:

Corrected Error generation. Describes the types of Corrected error that the fault generation feature of the node can generate.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	The fault generation feature of the node does not generate Corrected errors. ERR<n>PFGCTL.CE is RES0.
0b01	The fault generation feature of the node allows generation of a non-specific Corrected error, that is, a Corrected error that is recorded by setting ERR<n>STATUS.CE to 0b10. ERR<n>PFGCTL.CE is a read/write field. The values 0b10 and 0b11 in ERR<n>PFGCTL.CE are reserved.
0b11	The fault generation feature of the node allows generation of transient or persistent Corrected errors, that is, Corrected errors that are recorded by setting ERR<n>STATUS.CE to 0b01 or 0b11 respectively. ERR<n>PFGCTL.CE is a read/write field. The value 0b01 in ERR<n>PFGCTL.CE is reserved.

All other values are reserved.

If ERR<n>FR.FRX is 1 then ERR<n>FR.CE indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

DE, bit [5]

When the node supports this type of error:

Deferred Error generation. Describes whether the fault generation feature of the node can generate Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	The fault generation feature of the node does not generate Deferred errors. ERR<n>PFGCTL.DE is RES0.
0b1	The fault generation feature of the node allows generation of Deferred errors. ERR<n>PFGCTL.DE is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.DE indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

UEO, bit [4]

When the node supports this type of error:

Latent or Restartable Error generation. Describes whether the fault generation feature of the node can generate Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	The fault generation feature of the node does not generate Latent or Restartable errors. ERR<n>PFGCTL.UEO is RES0.
0b1	The fault generation feature of the node allows generation of Latent or Restartable errors. ERR<n>PFGCTL.UEO is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UEO indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

UER, bit [3]

When the node supports this type of error:

Signaled or Recoverable Error generation. Describes whether the fault generation feature of the node can generate Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	The fault generation feature of the node does not generate Signaled or Recoverable errors. ERR<n>PFGCTL.UER is RES0.
0b1	The fault generation feature of the node allows generation of Signaled or Recoverable errors. ERR<n>PFGCTL.UER is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UER indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

UEU, bit [2]

When the node supports this type of error:

Unrecoverable Error generation. Describes whether the fault generation feature of the node can generate Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	The fault generation feature of the node does not generate Unrecoverable errors. ERR<n>PFGCTL.UEU is RES0.
0b1	The fault generation feature of the node allows generation of Unrecoverable errors. ERR<n>PFGCTL.UEU is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UEU indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

UC, bit [1]

When the node supports this type of error:

Uncontainable Error generation. Describes whether the fault generation feature of the node can generate Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	The fault generation feature of the node does not generate Uncontainable errors. ERR<n>PFGCTL.UC is RES0.
0b1	The fault generation feature of the node allows generation of Uncontainable errors. ERR<n>PFGCTL.UC is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UC indicates whether the node supports this type of error.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

OF, bit [0]

When the node supports this flag:

Overflow flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.OF status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OF	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.OF according to the architecture-defined rules for setting the OF field. ERR<n>PFGCTL.OF is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.OF is set to ERR<n>PFGCTL.OF. This behavior replaces the architecture-defined rules for setting the OF bit. ERR<n>PFGCTL.OF is a read/write field.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

Accessing ERR<n>PFGF

This section shows the offset of ERR<n>PFGF in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGF is accessed in a fault injection group, see ‘RAS memory-mapped register views’ for the offset of ERR<n>PFGF.

ERR<n>PFGF can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x800 + (64 * n)`	ERR<n>PFGF

Accesses to this register are RO.

3.2.33 ERR<n>STATUS, Error Record <n> Primary Status Register, n = 0 - 65534

The ERR<n>STATUS characteristics are:

Purpose

When RAS System Architecture v2 is implemented, error record <n> might be one of the following:

- A continuation record containing more information about the error recorded in error record <n-1>. In this case, ERR<n>STATUS contains a subset of the values of a normal error record status register.
- A proxy for a different RAS agent. In this case, ERR<n>STATUS reports the status of the RAS agent.

Otherwise, ERR<n>STATUS contains status information for error record <n>, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a Requester.
- 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 reported 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:

- ERR<n>STATUS.{AV, V, MV} are valid bits that define whether error record <n> registers are valid.
- ERR<n>STATUS.{UE, OF, CE, DE, UET} encode the types of error or errors recorded.
- ERR<n>STATUS.{CI, ER, PN, IERR, SERR} are syndrome fields.

Configuration

ERRFRPFGF[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>STATUS, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

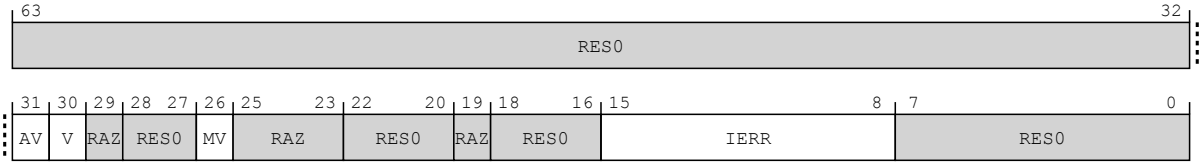
This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>STATUS are RES0.

Attributes

ERR<n>STATUS is a 64-bit register.

Field descriptions

When RAS System Architecture v2 is implemented, ERR<n>FR.ED == '00', and ERR<n>FR.ERT == '01':



Continuation record.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an additional address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == '1', this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. Additional syndrome has been recorded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == '1', this field resets to an architecturally UNKNOWN value.

- Otherwise, this field resets to '0'.

Access to this field is W1C.

Bit [29]

Reserved, RAZ.

Bits [28:27]

Reserved, RES0.

MV, bit [26]

When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
0b0	ERR<n>MISC<m> not valid.
0b1	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> 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 recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == '1', this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

Bits [25:23]

Reserved, RAZ.

Bits [22:20]

Reserved, RES0.

Bit [19]

Reserved, RAZ.

Bits [18:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED additional error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid 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.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

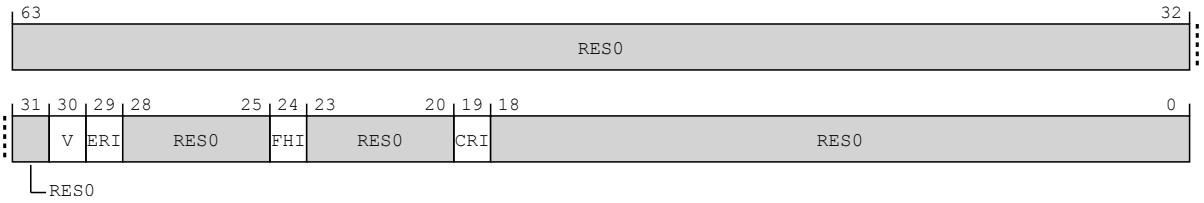
Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if all of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - the node that owns error record n does not implement the Common Fault Injection Model Extension
- Access to this field is UNKNOWN/WI if all of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - ERRPFGF[FirstRecordOfNode(n)].SYN == ‘0’
- Otherwise, access to this field is RW.

Bits [7:0]

Reserved, RES0.

When RAS System Architecture v2 is implemented, ERR<n>FR.ED == '11', and ERR<n>FR.ERT == '01':



Proxy for a RAS agent.

Bits [63:31]

Reserved, RES0.

V, bit [30]

RAS agent error status.

V	Meaning
0b0	RAS agent error status is not asserted.
0b1	RAS agent error status is asserted.

Access to this field is RO.

ERI, bit [29]

RAS agent Error Recovery condition.

ERI	Meaning
0b0	RAS agent error recovery condition is false.
0b1	RAS agent error recovery condition is true.

Access to this field is RO.

Bits [28:25]

Reserved, RES0.

FHI, bit [24]

RAS agent Fault Handling condition.

FHI	Meaning
0b0	RAS agent fault handling condition is false.
0b1	RAS agent fault handling condition is true.

Access to this field is RO.

Bits [23:20]

Reserved, RES0.

CRI, bit [19]

RAS agent criticial error condition.

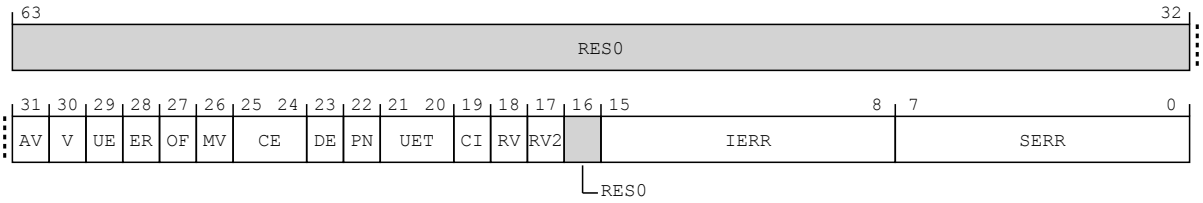
CRI	Meaning
0b0	RAS agent criticial error condition is false.
0b1	RAS agent criticial error condition is true.

Access to this field is RO.

Bits [18:0]

Reserved, RES0.

When RAS System Architecture v1p1 is implemented:



Normal record, from FEAT_RASSAv1p1.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == ‘1’, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. At least one error has been recorded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == ‘1’, this field resets to an architecturally UNKNOWN value.

- Otherwise, this field resets to '0'.

Access to this field is W1C.

UE, bit [29]

Uncorrected Error.

UE	Meaning
0b0	No errors have been detected, or all detected errors have been either corrected or deferred.
0b1	At least one detected error was not corrected and not deferred.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

ER, bit [28]

When in-band error responses can be returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, can signal an in-band error response to the Requester, causing this field to be set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
- Otherwise, access to this field is W1C.

When in-band error responses are never returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, can signal an in-band error response to the Requester, causing this field to be set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - ERR<n>STATUS.UE == ‘0’
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field is set to 1 when one of the following occurs:

- A Corrected error counter is implemented, an error is counted, and the counter overflows.
- ERR<n>STATUS.V was previously 1, a Corrected error counter is not implemented, and a Corrected error is recorded.
- ERR<n>STATUS.V was previously 1, and a type of error other than a Corrected error is recorded.

Otherwise, this field is unchanged when an error is recorded.

If a Corrected error counter is implemented, then:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.
- A direct write to this field that clears this field to zero might indirectly set the counter overflow flag to an UNKNOWN value.

OF	Meaning
0b0	Since this field was last cleared to zero, no error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed.
0b1	Since this field was last cleared to zero, at least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

MV, bit [26]

When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
0b0	ERR<n>MISC<m> not valid.
0b1	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> 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 recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == '1', this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

CE	Meaning
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 component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

DE, bit [23]

Deferred Error.

DE	Meaning
0b0	No errors were deferred.
0b1	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

PN, bit [22]

Poison.

PN	Meaning
0b0	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.

0b1	Uncorrected error or Deferred error recorded because a poison value was detected.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
- Otherwise, access to this field is W1C.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

UET	Meaning
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).

UER can mean either Signaled or Recoverable error, and UEO can mean either Latent or Restartable error.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - ERR<n>STATUS.UE == ‘0’
- Otherwise, access to this field is W1C.

CI, bit [19]

Critical Error. Indicates whether a critical error condition has been recorded.

CI	Meaning
0b0	No critical error condition.
0b1	Critical error condition.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

RV, bit [18]

When RAS System Architecture v2 is implemented:

Reset Valid. When ERR<n>STATUS.V is 1, indicating the error record is valid, this field indicates whether the error was recorded before or after the most recent Error Recovery reset.

RV	Meaning
0b0	If the error record is valid then one or more errors have been recorded after the last Error Recovery reset. This error or errors might have overwritten lower priority errors recorded before the last Error Recovery reset.
0b1	If the error record is valid then one or more errors were recorded before the last Error Recovery reset.

This field is set to 0 when an error is recorded and either the fault overwrites the error syndrome, or the error record was previously not valid.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to '1'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

RV2, bit [17]

When RAS System Architecture v2 is implemented:

Reset Valid 2. When ERR<n>STATUS.{V, RV} is {1, 1}, indicating the error record is valid and one or more errors were recorded before the last Error Recovery reset, this field indicates whether any lower severity errors have been recorded after the Error Recovery reset that did not overwrite the syndrome.

RV2	Meaning
0b0	If the error record is valid then one or more errors were recorded after the last Error Recovery reset that did not overwrite the error syndrome. This includes errors that did not overwrite a previously recorded error syndrome.
0b1	If the error record is valid then one or more errors were recorded before the last Error Recovery reset.

This field is set to 0 when an error is recorded, including when the fault does not overwrite a previously recorded syndrome.

The reset behavior of this field is:

- On a Error recovery reset, this field resets to '1'.

Access to this field is W1C.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid 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.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if all of the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension
  - ERR<n>STATUS.V == ‘0’
- Access to this field is UNKNOWN/WI if all of the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == ‘0’
  - ERR<n>STATUS.V == ‘0’
- Otherwise, access to this field is RW.

SERR, bits [7:0]

Architecturally-defined primary error code. 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.

SERR	Meaning
0x00	No error.
0x01	IMPLEMENTATION DEFINED error.
0x02	Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.
0x03	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
0x04	Assertion failure. For example, consistency failure.
0x05	Error detected on internal data path. For example, parity on ALU result.
0x06	Data value from associative memory. For example, ECC error on cache data.
0x07	Address/control value from associative memory. For example, ECC error on cache tag.
0x08	Data value from a TLB. For example, ECC error on TLB data.
0x09	Address/control value from a TLB. For example, ECC error on TLB tag.
0x0A	Data value from producer. For example, parity error on write data bus.
0x0B	Address/control value from producer. For example, parity error on address bus.
0x0C	Data value from (non-associative) external memory. For example, ECC error in SDRAM.

SERR	Meaning
0x0D	Illegal address (software fault). For example, access to unpopulated memory.
0x0E	Illegal access (software fault). For example, byte write to word register.
0x0F	Illegal state (software fault). For example, device not ready.
0x10	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are data registers.
0x11	Internal control register. For example, parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are control registers.
0x12	Error response from Completer of access. For example, error response from cache write-back.
0x13	External timeout. For example, timeout on interaction with another component.
0x14	Internal timeout. For example, timeout on interface within the component.
0x15	Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further.
0x16	Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further.
0x17	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
0x18	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
0x19	Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.
0x1A	Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code.

All other values are reserved.

The implemented set of valid 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.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

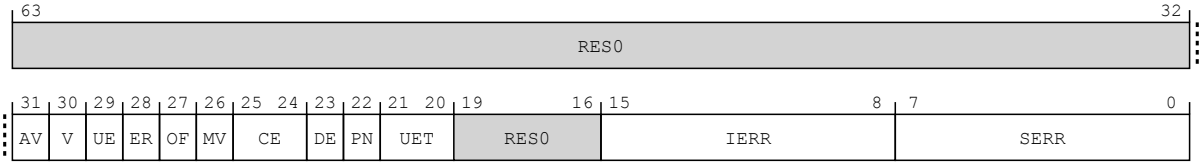
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if all of the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension
  - ERR<n>STATUS.V == ‘0’
- Access to this field is UNKNOWN/WI if all of the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == ‘0’
  - ERR<n>STATUS.V == ‘0’
- Otherwise, access to this field is RW.

Otherwise:



Normal record, when FEAT_RASSAv1p1 is not implemented.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RO if all of the following are true:
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. At least one error has been recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is W1C.

UE, bit [29]

Uncorrected Error.

UE	Meaning
0b0	No errors have been detected, or all detected errors have been either corrected or deferred.
0b1	At least one detected error was not corrected and not deferred.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == ‘0’, access to this field is UNKNOWN/WI.
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.OF == ‘1’
  - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write
- Otherwise, access to this field is W1C.

ER, bit [28]

When in-band error responses can be returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing both ERR<n>STATUS.{DE, UE} to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is W1C.

When in-band error responses are never returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

If this field is nonzero, then Arm recommends that software write 1 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.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - ERR<n>STATUS.UE == ‘0’
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to `0b0` in the same write
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field 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 field 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 ERR<n>STATUS.{UE, DE} != {0, 0}.
- A Corrected error counter is implemented, ERR<n>STATUS.{UE, DE} != {0, 0}, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this field 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`.

The IMPLEMENTATION DEFINED clearing of this field might also depend on the value of the other error status fields.

If a Corrected error counter is implemented, then:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.
- A direct write to this field that clears this field to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

OF	Meaning
0b0	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 field might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.
0b1	At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == ‘0’, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is W1C.

MV, bit [26]

When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
0b0	ERR<n>MISC<m> not valid.
0b1	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> 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 recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RO if all of the following are true:
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’

- ERR<n>STATUS.DE != ‘0’
- ERR<n>STATUS.DE is not being cleared to `0b0` in the same write

- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to `0b0` in the same write
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

CE	Meaning
`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 component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to `0b10` when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == ‘0’, access to this field is UNKNOWN/WI.
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.OF == ‘1’
  - ERR<n>STATUS.OF is not being cleared to `0b0` in the same write
- Otherwise, access to this field is W1C.

DE, bit [23]

Deferred Error.

DE	Meaning
`0b0`	No errors were deferred.
`0b1`	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == '0', access to this field is UNKNOWN/WI.
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.OF == '1'
  - ERR<n>STATUS.OF is not being cleared to `0b0` in the same write
- Otherwise, access to this field is W1C.

PN, bit [22]

Poison.

PN	Meaning
`0b0`	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.
`0b1`	Uncorrected error or Deferred error recorded because a poison value was detected.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing both ERR<n>STATUS.{DE, UE} to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == '0'
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == '00'
- Access to this field is RO if all of the following are true:
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == '00'
  - ERR<n>STATUS.CE != '00'
  - ERR<n>STATUS.CE is not being cleared to `0b00` in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == '0'
  - ERR<n>STATUS.DE != '0'
  - ERR<n>STATUS.DE is not being cleared to `0b0` in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != '0'
  - ERR<n>STATUS.UE is not being cleared to `0b0` in the same write
- Otherwise, access to this field is W1C.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

UET	Meaning
`0b00`	Uncorrected error, Uncontainable error (UC).

UET	Meaning
0b01	Uncorrected error, Unrecoverable error (UEU).
0b10	Uncorrected error, Latent or Restartable error (UEO).
0b11	Uncorrected error, Signaled or Recoverable error (UER).

UER can mean either Signaled or Recoverable error, and UEO can mean either Latent or Restartable error.

If this field is nonzero, then Arm recommends that software 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.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if any of the following are true:
  - ERR<n>STATUS.V == ‘0’
  - ERR<n>STATUS.UE == ‘0’
- Access to this field is RO if all of the following are true:
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is W1C.

Bits [19:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid 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.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if all of the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension
  - ERR<n>STATUS.V == ‘0’
- Access to this field is UNKNOWN/WI if all of the following are true:

- ERRPFGF[FirstRecordOfNode(n)].SYN == ‘0’
- ERR<n>STATUS.V == ‘0’

- Access to this field is RO if all of the following are true:
  - [ERR<n>STATUS.DE, ERR<n>STATUS.UE] == ‘00’
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to `0b00` in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to `0b0` in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to `0b0` in the same write
- Otherwise, access to this field is RW.

SERR, bits [7:0]

Architecturally-defined primary error code. 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.

SERR	Meaning
`0x00`	No error.
`0x01`	IMPLEMENTATION DEFINED error.
`0x02`	Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.
`0x03`	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
`0x04`	Assertion failure. For example, consistency failure.
`0x05`	Error detected on internal data path. For example, parity on ALU result.
`0x06`	Data value from associative memory. For example, ECC error on cache data.
`0x07`	Address/control value from associative memory. For example, ECC error on cache tag.
`0x08`	Data value from a TLB. For example, ECC error on TLB data.
`0x09`	Address/control value from a TLB. For example, ECC error on TLB tag.
`0x0A`	Data value from producer. For example, parity error on write data bus.
`0x0B`	Address/control value from producer. For example, parity error on address bus.
`0x0C`	Data value from (non-associative) external memory. For example, ECC error in SDRAM.
`0x0D`	Illegal address (software fault). For example, access to unpopulated memory.
`0x0E`	Illegal access (software fault). For example, byte write to word register.
`0x0F`	Illegal state (software fault). For example, device not ready.
`0x10`	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are data registers.
`0x11`	Internal control register. For example, parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are control registers.
`0x12`	Error response from Completer of access. For example, error response from cache write-back.
`0x13`	External timeout. For example, timeout on interaction with another component.
`0x14`	Internal timeout. For example, timeout on interface within the component.

SERR	Meaning
0x15	Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further.
0x16	Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further.
0x17	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
0x18	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
0x19	Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.
0x1A	Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code.

All other values are reserved.

The implemented set of valid 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.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is UNKNOWN/WI if all of the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension
  - ERR<n>STATUS.V == ‘0’
- Access to this field is UNKNOWN/WI if all of the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == ‘0’
  - ERR<n>STATUS.V == ‘0’
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.DE == ‘0’
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.CE != ‘00’
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE == ‘0’
  - ERR<n>STATUS.DE != ‘0’
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- Access to this field is RO if all of the following are true:
  - ERR<n>STATUS.UE != ‘0’
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is RW.

Accessing ERR<n>STATUS

ERR<n>STATUS.{AV, V, UE, ER, OF, MV, CE, DE, PN, UET, CI} are write-one-to-clear (W1C) fields, meaning writes of zero are ignored, and a write of one or all-ones to the field clears the field to zero. ERR<n>STATUS.{IERR, SERR} are read/write (RW) fields, although the set of implemented valid values is IMPLEMENTATION DEFINED. See also ERR<n>PFGF.SYN.

After reading ERR<n>STATUS, software must clear the valid fields in the register to allow new errors to be recorded. However, between reading the register and clearing the valid fields, a new error might have overwritten the register. To

prevent this error being lost by software, the register prevents updates to fields that might have been updated by a new error.

When RAS System Architecture v1.0 is implemented:

- Writes to ERR<n>STATUS.{UE, DE, CE} are ignored if ERR<n>STATUS.OF is 1 and is not being cleared to 0.
- Writes to ERR<n>STATUS.V are ignored if any of ERR<n>STATUS.{UE, DE, CE} are nonzero and are not being cleared to zero.
- Writes to ERR<n>STATUS.{AV, MV} and the ERR<n>STATUS.{ER, PN, UET, IERR, SERR} syndrome fields are ignored if the highest priority nonzero error status field is not being cleared to zero. The error status fields in priority order from highest to lowest, are ERR<n>STATUS.UE, ERR<n>STATUS.DE, and ERR<n>STATUS.CE.

When RAS System Architecture v1.1 is implemented, a write to the register is ignored if all of:

- Any of ERR<n>STATUS.{V, UE, OF, CE, DE} are nonzero before the write.
- The write does not clear the nonzero ERR<n>STATUS.{V, UE, OF, CE, DE} fields to zero by writing ones to the applicable field or fields.

Some of the fields in ERR<n>STATUS are also defined as UNKNOWN where certain combinations of ERR<n>STATUS.{V, DE, UE} are zero. The rules for writes to ERR<n>STATUS allow a node to implement such a field as a fixed read-only value.

For example, when RAS System Architecture v1.1 is implemented, a write to ERR<n>STATUS when ERR<n>STATUS.V is 1 results in either ERR<n>STATUS.V field being cleared to zero, or ERR<n>STATUS.V not changing. Since all fields in ERR<n>STATUS, other than ERR<n>STATUS.{AV, V, MV}, usually read as UNKNOWN values when ERR<n>STATUS.V is zero, this means those fields can be implemented as read-only if applicable.

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software performs the following sequence of operations in order:

1. Read ERR<n>STATUS and determine which fields need to be cleared to zero.
2. In a single write to ERR<n>STATUS:
   - Write ones to all the W1C fields that are nonzero in the read value.
   - Write zero to all the W1C fields that are zero in the read value.
   - Write zero to all the RW fields.
3. Read back ERR<n>STATUS after the write to confirm no new fault has been recorded.

Otherwise, these fields might not have the correct value when a new fault is recorded.

ERR<n>STATUS can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0x010 + (64 * n)`	ERR<n>STATUS

Accessible as follows:

- When ERR<n>STATUS.V != ‘0’, ERR<n>STATUS.V is not being cleared to `0b0` in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are RO.
- When ERR<n>STATUS.UE != ‘0’, ERR<n>STATUS.UE is not being cleared to `0b0` in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are RO.
- When ERR<n>STATUS.OF != ‘0’, ERR<n>STATUS.OF is not being cleared to `0b0` in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are RO.
- When ERR<n>STATUS.CE != ‘00’, ERR<n>STATUS.CE is not being cleared to `0b00` in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are RO.
- When ERR<n>STATUS.DE != ‘0’, ERR<n>STATUS.DE is not being cleared to `0b0` in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are RO.
- Otherwise, accesses to this register are RW.

3.2.34 ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

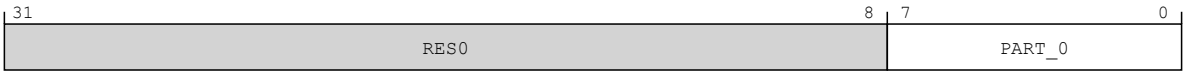
ERRPIDR0 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing ERRPIDR0

This section shows the offset of ERRPIDR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR0.

ERRPIDR0 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFE0	ERRPIDR0

Accesses to this register are RO.

3.2.35 ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

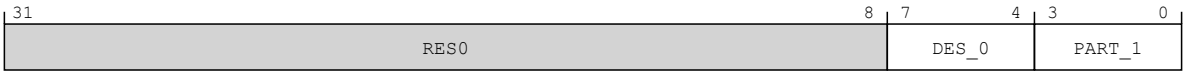
ERRPIDR1 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field has an IMPLEMENTATION DEFINED value.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

Access to this field is RO.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing ERRPIDR1

This section shows the offset of ERRPIDR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRPIDR1.

ERRPIDR1 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xFE4`	ERRPIDR1

Accesses to this register are RO.

3.2.36 ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

ERRPIDR2 is implemented only as part of a memory-mapped group of error records.

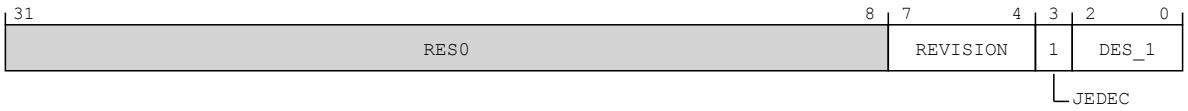
Implementation of this register is OPTIONAL.

Attributes

ERRPIDR2 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with ERRPIDR2.REVISION being the most significant part and ERRPIDR3.REVAND the least significant part. When a component is changed, ERRPIDR2.REVISION or ERRPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. ERRPIDR3.REVAND should be set to `0b0000` when ERRPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`

Access to this field is RO.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

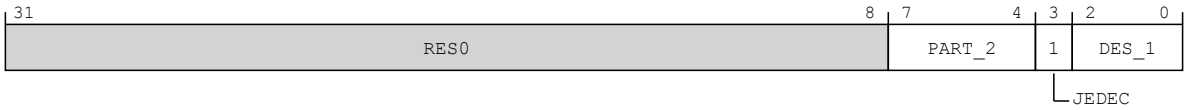
This field has an IMPLEMENTATION DEFINED value.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

Access to this field is RO.

When the component uses a 16-bit part number:



Bits [31:8]

Reserved, RES0.

PART_2, bits [7:4]

Part number, bits [15:12].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`

Access to this field is RO.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field has an IMPLEMENTATION DEFINED value.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

Access to this field is RO.

Accessing ERRPIDR2

This section shows the offset of ERRPIDR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRPIDR2.

ERRPIDR2 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFE8	ERRPIDR2

Accesses to this register are RO.

3.2.37 ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

ERRPIDR3 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRPIDR3 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with ERRPIDR2.REVISION being the most significant part and ERRPIDR3.REVAND the least significant part. When a component is changed, ERRPIDR2.REVISION or ERRPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. ERRPIDR3.REVAND should be set to `0b0000` when ERRPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

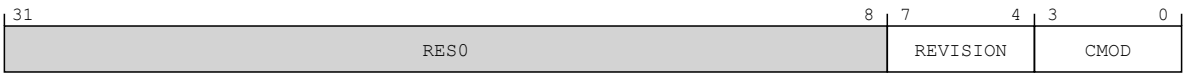
This field has an IMPLEMENTATION DEFINED value.

For any two components with the same Unique Component Identifier:

- If ERRPIDR3.CMOD is zero in both components, then the components are identical.
- If ERRPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If ERRPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

Access to this field is RO.

When the component uses a 16-bit part number:



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component revision. When a component is changed, ERRPIDR3.REVISION is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

This field has an IMPLEMENTATION DEFINED value.

For any two components with the same Unique Component Identifier:

- If ERRPIDR3.CMOD is zero in both components, then the components are identical.
- If ERRPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If ERRPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

Access to this field is RO.

Accessing ERRPIDR3

This section shows the offset of ERRPIDR3 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see ‘RAS memory-mapped register views’ for the offset of ERRPIDR3.

ERRPIDR3 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	`0xFEC`	ERRPIDR3

Accesses to this register are RO.

3.2.38 ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

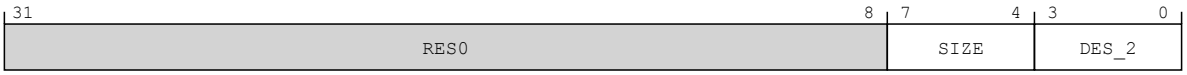
ERRPIDR4 is implemented only as part of a memory-mapped group of error records.

Implementation of this register is OPTIONAL.

Attributes

ERRPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

When RAS System Architecture v2 is implemented:

Size of the component.

SIZE	Meaning
0b0000	FEAT_RASSA_4KB is implemented.
0b0010	FEAT_RASSA_16KB is implemented.
0b0100	FEAT_RASSA_64KB is implemented.

All other values are reserved.

Otherwise:

Size of the component.

SIZE	Meaning
0b0000	One of the following is true: • The component uses a single 4KB block. • The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

SIZE	Meaning
0b0001..0b1111	The component occupies $2^{ERRPIDR4.SIZE}$ 4KB blocks.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

This field has an IMPLEMENTATION DEFINED value.

Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4.

Access to this field is RO.

Accessing ERRPIDR4

This section shows the offset of ERRPIDR4 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR4.

ERRPIDR4 can be accessed through the memory-mapped interface:

Component	Offset	Instance
RAS	0xFD0	ERRPIDR4

Accesses to this register are RO.

Glossary

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 Can refer to either a *corrected error* or the *Corrected* component error state..

Completer An agent in a computing system that responds to and completes a transaction initiated by a Requester.

Component error state A syndrome value recorded when a RAS node records that an error has been detected, that informs software of what actions the component has taken, and directs software recovery.

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.

Corrected error An error that is detected by hardware and that hardware has corrected.

DE Can refer to either a *deferred error* or the *Deferred* component error state.

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 Uncorrected Error A detected error that has not been be corrected and causes failure.

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.

DUE Detected Uncorrected 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 log Historical data recorded about errors, usually by software.

Error propagation Passing an error from a producer to a consumer.

Error record Data recorded about an error, usually by hardware.

Exception An exception handles an event. For example, an exception could handle an external interrupt or an undefined instruction.

External abort Either:

- 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.
- A type of exception in the Arm architecture, generated when consuming an in-band error response.

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.

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 A component or unit in a system that can be replaced without return to base.

FIT Failure-in-Time.

FRU Field Replaceable Unit.

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.

Infected Being in error.

Interrupt 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 error or 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.

MSI Message Signaled Interrupt.

Normal memory Used for bulk memory operations. Hardware might speculatively read these locations.

PCIe Peripheral Component Interconnect Express.

PE Processing Element.

Peripheral Component Interconnect Express (PCI Express or PCIe) A high-speed serial computer expansion bus standard maintained and developed by the PCI Special Interest Group.

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 be treated as a detected error.

PPI Private Peripheral Interrupt.

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 conforms with the behaviors described in the corresponding Arm Architecture Reference Manual.

Propagated See Error propagation.

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 state The state of a component when it has contained an error, and that error must be corrected to allow the correct operation of the system or smaller parts of the system to continue.

Reliability Continuity of correct service.

Requester An agent in a computing system that initiates transactions.

Restartable state The state of a component when it has contained error, and the error 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.

Service failure mode A mode entered to reduce the severity of an error.

Serviceability The ability to undergo modifications and repairs.

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.

Transient fault A fault that is not persistent.

UE Can refer to either an *uncorrected error* or the *Uncorrected* component error state.

Uncontained or uncontainable error An error that has been, or might have been, silently propagated.

Undetected error or undetected fault See Latent error or latent fault.

Unrecoverable state The state of a component that has contained an error, but the state is not recoverable and 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.