Arm® Architecture Reference Manual Supplement

Reliability, Availability, and Serviceability (RAS), for Armv8-A

ARM DDI 0587 D.b-00bet1

Arm RAS Supplement

Release information

Date	Version	Changes
2021/Jan/25	D.b	• Updated v8.6 Beta release.
2020/Jul/22	D.a	• Initial v8.6 Beta release, with rewrite of the RAS supplement.
2019/Jul/01	C.b	• Updated v8.4 release.
2018/Oct/01	C.a	• Initial v8.4 EAC release.
2017/Dec/01	B.a	• Updated EAC release.
2017/Sep/01	B	• EAC release.
2017/Mar/01	A	• First issue.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Contents

Arm RAS Supplement

Preface

Document status

Beta release.

Beta quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release will be sufficiently stable and committed for initial product development.

The recipient can expect some changes to the Beta quality released material.

In case of any apparent discrepancy or missing information, please contact Arm Limited.

About this book

This manual describes the *Armv8-A RAS Extension* and the *RAS System Architecture*.

Using this book

This manual is intended to be read in conjunction with [1].

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

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

`monospace`

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

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

SMALL CAPITALS

Used in body text for terms, such as IMPLEMENTATION DEFINED, that have specific technical meanings described in the Arm Architecture Reference Manual.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be a cross-reference to another location within the document, or a URL such as http://developer.arm.com.

Numbers

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

Pseudocode descriptions

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

Assembler syntax descriptions

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

Rules-based writing

This specification consists of a set of individual *content items*. Content items are classified into the following types:

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

Rules are normative statements. An implementation which is compliant with this specification must conform to all of the Rules in this specification.

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

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 purely as an aid to understanding this specification.

Content item classes

Rule

A Rule is a statement which either

- describes the behaviour of a compliant implementation, or
- defines concepts or terminology.

A Rule is identified by the letter R.

Information

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

An Information statement is identified by the letter I.

Rationale

A Rationale statement explains why the specification was specified as it was.

A Rationale statement is identified by the letter X.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is identified by the letter U.

Software usage

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

A Software usage statement is identified by the letter S.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001*, *0002*, . . .).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

R	This is a Rule.
R_{X001}	This is a Rule with an identifier.
X	This is a Rationale statement.
I	This is an Information statement.
U	This is an Implementation note.
S	This is a Software usage statement.

Additional reading

This section lists publications by Arm and by third parties.

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

[1] *Arm® Architecture Reference Manual for ARMv8-A architecture profile*. (ARM DDI 0487) Arm Limited.

[2] *Basic Concepts and Taxonomy of Dependable and Secure Computing*. Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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

- The title (Arm RAS Supplement).
- The number (ARM DDI 0587 D.b-00bet1).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Progressive terminology statement

Arm values inclusive communities.

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

Previous issues of this document included terms that can be offensive. We have replaced these terms.

See Release information.

If you find offensive terms in this document, please contact terms@arm.com.

Chapter 1
Introduction to RAS

I_LMHPD *Reliability, Availability, Serviceability* (RAS) are three aspects of the *dependability* of a system:

- *Reliability*, the continuity of correct service.
- *Availability*, the readiness for correct service.
- *Serviceability*, 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.

1.1 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.

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

R_{NSPJY}
- A *failure* is the event of deviation from correct service. This includes data corruption, data loss, and service loss.

R_{SCKWX}
- An *error* is the deviation from correct service. An incorrect value that has an error is *corrupt*.

R_{YRDDR}
- 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 Extension and RAS System Architecture primarily address errors produced from hardware faults. These fall into two main areas:

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

1.2 General taxonomy of errors

1.2.1 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*.

1.2.2 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.

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:

R_{XDHGD}
- A corrupt value is passed from producer to consumer.

R_{ZCNXB}
- A transaction or other operation occurs that should not have occurred.

R_{CFZKP}
- A transaction or other operation that should have occurred does not occur.

R_{MZVNK}
- A loss of uniprocessor semantics or any other loss of coherency in a multiprocessor coherent system is observed.

R_{KCDXV}
- 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.

The service interface for a transaction might include means to *signal* that the transaction is propagating:

R_{VVFYS}
- A detected error.

R_{CSVRC}
- 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.

I_{XYVCP} For example, a partial write to a protection granule that 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.

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

I_{YZTDY}
- 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.

I_{VQZPT}
- 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 SDC Failure-in-Time *(FIT)* rate.

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

1.2.3 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.

1.2.4 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 or containable at the component, it treats it as uncontainable at the component.

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

Note

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

1.3 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.

1.3.1 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.

R_{JVLNC} A fault that is removed is a *corrected error* and might be recorded and generate a fault handling interrupt, but it is not propagated. This means that it is not consumed and does not cause service failure.

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 any fault removal schemes, including ECC.

1.3.2 Error handling and recovery

R_{XPLVT} A fault that is not removed gives rise to an *uncorrected error*.

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

Error recovery methods include:

I_{DCGYX}
- *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 *Processing element* (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.

I_{YLMTV}
- Preventing further propagation of the error, that is *containing* the error. In particular, preventing silent propagation of the error.

I_{RFLCM}
- 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.

1.3.3 Fault handling

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

Fault handling methods include:

I_{GGCDN}
- *Predictive Failure Analysis* (PFA), using information recorded by hardware to trigger pre-emptive 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.

See also:

- *Standard error record*

Chapter 2
Armv8-A RAS Extension

I_{LNLHW} The *Reliability, Availability, Serviceability* (RAS) Extension is identified as *FEAT_RAS*.

I_{FNVKV} The RAS Extension is a mandatory extension to the Armv8.2 architecture, and it is an optional extension to the Armv8.0 and Armv8.1 architectures.

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

I_{LBKPL} The RAS Extension extends the exception syndrome registers to include fields that allow the *Processing element* (PE) to report a PE error state when an error exception is taken.

I_{DWKZS} The RAS Extension adds the Error synchronization event and Error Synchronization Barrier instruction, `ESB`.

I_{THGHB} The RAS Extension defines System registers that are specific to RAS, including to access optional error records defined by the *RAS System Architecture*. The System register instructions are described in [1]. The format of the error record registers is defined in *Error record System register view*.

I_{KPYCD} The *FEAT_IESB* feature provides controls to insert an implicit Error synchronization event at exception entry and exception return.

I_{KJCLV} The *FEAT_RASv1p1* feature extends the RAS System registers to include support for *RAS System Architecture* v1.1.

I_{GWRVK} The *FEAT_DoubleFault* feature provides EL3 controls to change the routing of synchronous External abort exceptions and treat SError interrupts as nonmaskable. FEAT_DoubleFault is defined in [1].

R_{YWXWL} The RAS Extension does not prescribe the level of reliability, availability, and serviceability in the PE. The RAS features that the PE includes, for example to detect, correct, contain, or defer errors, are IMPLEMENTATION DEFINED. The RAS Extension defines a framework for building RAS features in a PE.

2.1 PE error handling

2.1.1 PE error detection

I_{KRYQW} When a PE accesses memory or other state, an error might be detected in that memory or state, and corrected, deferred, or signaled to the PE as a detected error with an *in-band error response*.

Note

An error might also be signaled to a PE by means other than an in-band error response. See R_{FNVVJ}.

The response from memory or other state is defined by *Detecting and consuming errors* in the *RAS System Architecture*:

I_{DWWQJ}
- When an error is detected by a component on a read or a cache maintenance operation from the PE:
  - If the error can be corrected, it is corrected and corrected data is returned.
  - If the error cannot be corrected and can be deferred, it is deferred.
  - Otherwise, if enabled at the component, the detected error is signaled to the PE as an in-band error response.

  The component might record the error and generate a fault handling interrupt and/or error recovery interrupt.

I_{BKVQP}
- When an error is detected by a component consuming a write from the PE:
  - If the error can be corrected, it is corrected.
  - If the error cannot be corrected and can be deferred, it is deferred to the consumer. For example, by poisoning the location being written.
  - If enabled at the consumer, the detected error is signaled to the PE as an in-band error response.
  - If enabled at the consumer, the consumer generates an error recovery interrupt.

I_{VRYFF} If the component implements the RAS System Architecture, its behavior is defined by *RAS System Architecture*, and depends on the nature of the error and IMPLEMENTATION DEFINED properties of the component. In each of these cases, the component might be a part of the processor, such as a cache, or might be outside of the processor.

The component might also report the error to a RAS System Architecture node, which records the error and might generate one or more of a fault handling interrupt, error recovery interrupt, or critical error interrupt depending on the features and configuration of the node.

See also *Other errors*.

Note

An in-band error response is sometimes referred to as an *External abort*. To avoid confusion with the External abort *exception*, this manual uses in-band error response to describe the response to the PE for a memory access.

See *In-band error response signaling (external aborts)*.

R_{QTRKF} The features that the system and PE include to detect, correct, or defer errors are IMPLEMENTATION DEFINED.

R_{WLTPV} The size of the protection granule for any implemented error detection mechanism in memory is IMPLEMENTATION DEFINED.

I_{JJJGW} A system might implement multiple error detection mechanisms with differing protection granule sizes.

R_{FNGVW} The mechanism for clearing an error or poison from a memory protection granule is IMPLEMENTATION DEFINED, and it is IMPLEMENTATION DEFINED whether any such mechanism exists.

Note

For some systems, a single-copy atomic write of at least the whole protection granule can reset the state of the granule and clear any error or poison. In other systems, a `DC ZVA` operation might also clear the error. However, the protection granule might be larger than the `DC ZVA` block size and/or the largest single-copy atomic access that the PE can perform.

Systems might require software to stop using the protection granule, for example by not using the physical page containing the granule, until the system can be purged of errors, for example at a system reset. The architecture does not set any limit on the size of a protection granule and it might be larger than a translation granule.

Any mechanism for purging the system of errors is also IMPLEMENTATION DEFINED.

2.1.2 PE error propagation

I_{NTXKV} The program-visible architectural state of the PE, referred to as the *PE state*, includes:

- General-purpose, SIMD&FP, and SVE registers.
- System registers.
- Special-purpose registers.
- PSTATE.

R_{XMBNW} An error is *consumed by the PE* by any of the following:

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

I_{HVFKW} For a PE, *Error propagation* applies to the propagation of detected errors by the PE between the PE state, and any other PE state or memory.

Note

Memory includes structures that cache the contents of memory, such as an instruction cache, data cache, or TLB.

An error is *propagated by the PE* by one or more of the following occuring that would not have been permitted to occur had the fault not been activated:

R_{DQTHR}
- Consumption of the corrupt value by any instruction, propagating the error to the target(s) of the instruction. This includes:
  - A store of a corrupt value.
  - A write of a corrupt value to a System register, Special-purpose register, or PSTATE. Infecting a System register state might mean that the PE generates transactions that would not otherwise be permitted.

R_{JKPCK}
- Any operation occuring that should not have occurred, including:

  - A load, translation table walk, or instruction fetch that would not have been permitted, including those from hardware speculation or prefetching.
  - A store to an incorrect address or a store that would not have been made or not permitted.
  - A direct or indirect write to a Special-purpose or System register that would not have been made or not permitted.
  - Assertion of any signal, such as an interrupt, that would not have been asserted.

R_{LLKVP}
- Any operation not occurring that should have occurred.

R_{PZNNG}
- Causing the PE to take an imprecise exception, other than an error exception in response to the error itself. See the section *Definition of a precise exception* in [1].

R_{PMMDF}
- The PE discarding data that it holds in a modified state.

R_{DDFXY}
- Any other loss of required uniprocessor semantics, ordering, or coherency.

R_{NQDWB} The error propagated by the PE is *silently propagated by the PE* only if all of the following are true:

- The propagation is not part of the required operation of the PE in taking an error exception generated by the error.
- The propagation is not part of the required operation of the PE executing an `ESB` instruction that synchronizes the error.
- The error is not signaled to the consumer as a detected error or deferred error.
- Any of the following are true:
  - The corrupt value is held in other than the general-purpose, SIMD&FP, or SVE registers.
  - The error is propagated by an instruction in program order before either taking an error exception generated by the error or executing an `ESB` instruction that synchronizes the error, and is propagated to outside of the general-purpose, SIMD&FP, or SVE registers.
  - The error is propagated other than by an instruction that consumes the corrupt value as an input operand but otherwise behaves correctly.

Note

This means that after taking the error exception generated by the error, or an `ESB`, propagating an error by, for example, storing it to memory, is not considered as silent propagation of the error by the PE.

For example, the PE takes an error exception in response to a load that returns a corrupt value to a general-purpose register. The error is not silently propagated to outside of the general-purpose registers before the error exception is taken. However:

- Taking the error exception causes the ESR_ELx, ELR_ELx, and SPSR_ELx registers to be updated. This is part of the required operation of the PE.
- After taking the error exception, software stores the contents of the general-purpose register to memory, and this is not signaled to memory as a deferred error. This happens in program order after the exception is taken.

Neither of these actions are considered silent propagation of the error by the PE.

R_{DTRFQ} The features that a PE includes to prevent silent propagation of an error are IMPLEMENTATION DEFINED.

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

2.1.3 Other errors

I_{KRQMR} The RAS Extension deals mostly with errors detected by components outside of the PE, such as memory, and consumed by the PE.

Other errors might be detected from within the processor itself. If these are not errors in the PE state they might be treated like errors detected by another component.

For example:

- A processor cache detects an error in the cache state that cannot be corrected. The cache can be treated as a component outside the PE. If the error is detected in dirty cache data being evicted from the cache when the PE makes an access, it might be deferred by the cache writing poison in the evicted cache data. If the PE is performing a partial write that does not completely overwrite the protection granule, it might be deferred by the cache writing poison to the cache location, and/or evicting the cache line with poison. Deferring the error means the error is not consumed by the PE. Otherwise, the cache component generates the in-band error response to the PE.
- A processor detects a corrupt or poisoned value being returned from memory that is not being signaled as an in-band error response and cannot be corrected or deferred. For example in response to an non-cacheable read or a cache refill. The interface to memory can be treated as a component outside the PE. The memory interface component generates the in-band error response to the PE.

In each of these cases, the *component* reports these errors to a *RAS System Architecture* node that implements error records and records the errors, and might generate one or more of a fault handling interrupt, error recovery interrupt, or critical error interrupt depending on the features and configuration of the node.

I_{VNTWD} An example implementation might include error detection logic within the PE state itself. When the PE detects an error in the PE state, the instruction that uses that state consumes the error, and the PE generates an IMPLEMENTATION DEFINED SError interrupt exception. See R_{FNVVJ}.

In this case, the processor that implements the PE includes a *RAS System Architecture* node that implements error records that record these errors.

I_{JRQDM} An example implementation might support poisoning within the PE state. When the PE consumes a deferred error, for example a poisoned value, from memory into the PE state, the PE state becomes poisoned. Subsequent operations that read the poisoned value can continue to defer the error by poisoning the result of the operation.

However, if the PE attempts to execute an operation that reads the poisoned value and cannot defer the error further, the PE generates an IMPLEMENTATION DEFINED SError interrupt exception. See R_{FNVVJ}.

In this case, the processor that implements the PE includes a *RAS System Architecture* node that implements error records that record these errors.

I_{JHQVK} Components outside of the PE might detect errors that are not consumed by the PE. These components might report such errors to a PE using error recovery interrupts.

R_{XJNNT} For implementations that include the Statistical Profiling Extension, the Statistical Profiling Extension behaves like a separate component.

I_{MJQQZ} Errors from software faults are outside the scope of the RAS Extension.

See also:

- *RAS System Architecture*
- *Software faults*

2.2 Generating error exceptions

R_{VJRKF} An *error exception* is generated when a detected error is signaled to the PE as an in-band error response to an architecturally-executed memory access or cache maintenance operation. This includes any explicit data access, instruction fetch, translation table walk, or hardware update to the translation tables made by an architecturally-executed instruction.

I_{PJHZS} Error exception is a term used in this manual to describe a collection of exception types. See *Taking error exceptions* for more information.

R_{MBNBH} It is IMPLEMENTATION DEFINED whether an error exception can be generated for an error that is consumed by hardware speculation or prefetching by a PE, but that is not committed to the architecturally visible state of the PE.

R_{SHKJB} It is IMPLEMENTATION DEFINED whether an error exception can be generated for a detected error that is deferred.

R_{GVWJD} It is IMPLEMENTATION DEFINED whether an error exception can be generated for a detected error that is corrected.

R_{FNVVJ} An SError interrupt exception can also be generated for IMPLEMENTATION DEFINED causes.

For example, when an error is detected and neither corrected nor deferred, and signaled to the PE by means other than an in-band error response, or when an error detected by the PE in the PE state or in the result of a calculation.

2.3 Taking error exceptions

R_{VXFYS} If FEAT_DoubleFault is implemented, an error exception is taken as a synchronous External abort exception for all non-speculative:

- Instruction fetches.
- Translation table walks and hardware updates of translation tables on instruction fetches.

It is IMPLEMENTATION DEFINED whether an error exception is taken as a synchronous External abort exception or as an asynchronous SError interrupt exception for each non-speculative:

- If FEAT_DoubleFault is not implemented, instruction fetch.
- Explicit access to memory made by an instruction.
- Cache maintenance operation.
- Translation table walk or hardware update of translation tables, other than for on an instruction fetch when FEAT_DoubleFault is implemented.
- If FEAT_MTE is implemented, access to an Allocation Tag in memory made by an instruction.

All error exceptions other than those explicitly mentioned in this rule are taken as an asynchronous SError interrupt exception.

R_{WFNJG} When an error exception is taken as an asynchronous SError interrupt exception, the exception is taken in finite time.

R_{BCXKN} When any of the following exceptions are taken, the PE records the *PE error state* in the exception syndrome register:

- A synchronous External abort taken to AArch64 state.
- An SError interrupt exception taken to either AArch32 or AArch64 state.

See *PE error state classification*.

R_{TYVYR} When a synchronous External abort is taken to AArch32 state, the PE does not record the PE error state.

I_{GGCQQ} The exception type and target execution state determines the set of PE error state values the PE can record. See *PE error state recording in the exception syndrome*.

I_{WSYXB} The recorded PE error state informs software whether it can recover execution and, if so, whether any action by the recovery software to locate and repair the error is necessary first.

I_{NFDSM} Software is only able to successfully *recover execution* and make progress from a restart address for the exception by executing an Exception Return instruction to branch to the instruction at this restart address if all of the following are true:

- The error has not been silently propagated by the PE.
- At the point when the Exception Return instruction is executed, the PE state and memory system state are consistent with the PE having executed all of the instructions up to but not including the instruction at the restart address, and none afterwards. That is, at least one of the following *restart conditions* is true:
  - The error has been not architecturally consumed by the PE and infected the PE state.
  - Executing the instruction at the restart address will not consume the error and will correct any corrupt state by overwriting it with the correct value or values.

R_{DCKHJ} On taking an error exception, the PE determines that software is able to recover execution at the point where the exception is taken, with no additional action from software, if and only if all of the following are true:

- The error has not been silently propagated by the PE.
- The restart conditions are met because all of the following are true:

  - Either the error does not remain latent or executing the instruction at the restart address will not consume the error and will correct any corrupt PE state.
  - The restart address is the preferred return address for the exception.

- The PE has not elected to determine that software is not able to recover execution, and has not elected to determine that software is able to recover execution if software takes action to locate and repair the error.

R_{JBHWY} On taking an error exception, the PE determines that software is able to recover execution if software takes action to *locate and repair* the error, to get the PE state and memory system state into this consistent state before attempting recovery, if and only if all the following are true:

- The error has not been silently propagated by the PE.
- The restart conditions can be met because the restart address is the preferred return address for the exception and at least one of the following is true:
  - The error remains latent and executing the instruction at the restart address will access the corrupt state. If the error is removed then executing the instruction at the restart address will correct any corrupt PE state and/or corrupt memory state. For example, the instruction at the restart address is a load that will consume the error and corrupts PE state.
  - The error does not remain latent and the PE has elected to determine that software is able to recover execution if software takes action to locate and repair the error.
  - Executing the instruction at the restart address will not consume the error and the PE has elected to determine that software is able to recover execution if software takes action to locate and repair the error.
- The PE has not elected to determine that software is not able to recover execution.

R_{GJQWN} On taking an error exception, the PE determines that software is not able to recover execution if and only if one or more of the following are true:

- The error has been silently propagated by the PE.
- The restart conditions cannot be met even if software takes action to locate and repair the error. This is because at least one of the following is true:
  - The error remains latent and executing the instruction at the restart address will consume the error and corrupt PE state. Either the error cannot be removed or executing the instruction at the restart address will not correct any corrupt PE state.
  - The restart address is not the preferred return address for the exception.
- The PE has elected to determine that software is not able to recover execution.

I_{XMCCR} That the PE determines that software is able to recover execution if software takes action to locate and repair the error does not mean that software can locate and repair. For example, the error in memory might be one which cannot be located or cannot be repaired.

Note

Error recovery software might instead make the PE state and memory system state consistent with an *alternative execution* of the program.

For example, if the error is located in a clean page of memory and the error exception is generated by a load from the location infected with the error, then software might be able to repair the error by:

- Reloading the page from a backing store. This makes the memory system state consistent with the uncorrupted view. Executing the instruction at the restart address will load the uncorrupted value into the PE state.
- Invalidating the clean page and marking it page as inaccessible. Executing the instruction at the restart

address will result in a Translation fault being generated when the program tries to access the page. The target of the load will contain an UNKNOWN value, which is permitted by the architecture. The MMU fault handler can then reload the page from the backing store, as it would for a page that has not been previously accessed or has been paged out.

Either approach might result in the virtual address to physical address mapping for the page being changed by software, meaning the memory system state is not consistent with the previously executed instructions. However, the memory system state is consistent with a valid alternative view of the execution of the program that allows software to recover execution.

I_{RHPPV} A PE might include additional IMPLEMENTATION DEFINED mechanisms to aid software locate and repair the error.

If software has to use IMPLEMENTATION DEFINED mechanisms to locate and repair the error, then the PE reports that it has determined that software is not able to recover execution. The PE might use IMPLEMENTATION DEFINED additional syndrome registers to report that software is able to recover execution if software takes action to locate and repair the error using the IMPLEMENTATION DEFINED mechanisms.

2.3.1 PE error state recording in the exception syndrome

R_{WPKYM} When an asynchronous SError interrupt exception is taken to AArch64 state, the PE records the PE error state in the ESR_ELx exception syndrome register as the applicable one of:

- Uncontainable (UC).
- Unrecoverable state (UEU).
- Recoverable state (UER).
- Restartable state (UEO).
- Corrected (CE).
- Uncategorized error.
- IMPLEMENTATION DEFINED syndrome.

I_{SDDLL} When an asynchronous SError interrupt exception is taken to AArch64 state:

- Uncategorized error is recorded by setting ESR_ELx.ISS to zero. This includes setting ESR_ELx.IDS and ESR_ELx.DFSC to zero.
- IMPLEMENTATION DEFINED syndrome is recorded by setting ESR_ELx.IDS to `0b1`. The remainder of the ESR_ELx.ISS syndrome is IMPLEMENTATION DEFINED.

Other values for the PE error state are recorded in ESR_ELx.AET, by setting ESR_ELx.IDS to `0b0` and ESR_ELx.DFSC to the applicable nonzero fault status code, indicating ESR_ELx.AET is valid.

R_{FKHHF} When a synchronous External abort exception is taken to AArch64 state, the PE records the PE error state in ESR_ELx.SET as the applicable one of:

- Uncontainable (UC).
- Recoverable state (UER).
- Restartable state (UEO).

Other values for the PE error state are not supported by synchronous External abort exceptions taken to AArch64 state.

R_{PWKBL} When an asynchronous SError interrupt exception is taken to AArch32 state, the PE records the PE error state in DFSR.AET or HSR.AET as appropriate, as the applicable one of:

- Uncontainable (UC).
- Unrecoverable state (UEU).
- Recoverable state (UER).
- Restartable state (UEO).

Other values for the PE error state are not supported by asynchronous SError interrupt exceptions taken to AArch32 state.

I_{QVVSM} Table 2.1 summarizes the supported PE error state syndrome values for each type of error exception.

Table 2.1: Summary of error exception types and supported PE error state syndrome values

PE error state	External abort to AArch64 state	SError interrupt to AArch64 state	External abort to AArch32 state	SError interrupt to AArch32 state
Recorded in:	ESR_ELx.SET	ESR_ELx.AET	No syndrome	DFSR.AET
Uncategorized error	No	Yes (ISS==0)	-	No
IMPLEMENTATION DEFINED syndrome	No	Yes (IDS==1)	-	No
Uncontainable (UC)	Yes (0b10)	Yes (0b000)	-	Yes (0b00)
Unrecoverable state (UEU)	No	Yes (0b001)	-	Yes (0b01)
Recoverable state (UER)	Yes (0b00)	Yes (0b011)	-	Yes (0b11)
Restartable state (UEO)	Yes (0b11)	Yes (0b010)	-	Yes (0b10)
Deferred (DE)	No	No	-	No
Corrected (CE)	No	Yes (0b110)	-	No

2.3.2 PE error state classification

I_{CCKWK} The PE determines which PE error state to record based on the following criteria:

- The PE error state syndrome values supported by the type of error exception being taken. See *PE error state recording in the exception syndrome*.
- The following implementation-specific properties and behaviors of the PE on taking the exception:
  - Whether the error has been silently propagated by the PE.
  - Whether the PE determines that software is able to recover execution at the point where the exception is taken.
  - If the PE determines that software can recover execution, whether software needs locate and repair the error before attempting to recover. If software does not locate and repair the error, then attempting to recover execution might cause the error exception to be generated again.
  - If the PE determines that software cannot recover execution, whether the error is synchronized by Error synchronization events.
- Whether the implementation elects to record the PE error state as another state. The PE only does this when the criteria for the other, recorded state are met. The conditions under which the PE elects to record the PE error state as another state are IMPLEMENTATION DEFINED.

The recorded PE error state is defined by the rules in this section.

R_{QKZLB} If and only if all of the following are true, then on taking an error exception the PE error state is recorded as *Uncontainable (UC)*:

- One or more of the following are true:
  - The error has been silently propagated by the PE.
  - The PE determines that software is not able to recover execution from the preferred return address of the exception and the error is not synchronized by Error synchronization events.
  - The PE determines that software is not able to recover execution from the preferred return address of the exception and the error exception is taken as a synchronous External abort to AArch64 state.

(That is, the type of error exception does not support reporting the PE error state as Unrecoverable state (UEU).)
  - The implementation has elected to record the PE error state as Uncontainable (UC).
- The error exception is not taken as a synchronous External abort to AArch32 state.
- The implementation has not elected to record the PE error state as IMPLEMENTATION DEFINED syndrome or Uncategorized error, or the type of error exception does not support reporting the PE error state as IMPLEMENTATION DEFINED syndrome or Uncategorized error.

R_{QGNYD} If and only if all of the following are true, then on taking an error exception the PE error state is recorded as *Unrecoverable state (UEU)*:

- The error has not been silently propagated by the PE.
- The error exception is taken as an SError interrupt exception.
- One or more of the following are true:
  - The PE determines that software is not able to recover execution from the preferred return address of the exception and the error is synchronized by Error synchronization events.
  - The implementation has elected to record the PE error state as Unrecoverable state (UEU).
- The implementation has not elected to record the PE error state as Uncontainable (UC), IMPLEMENTATION DEFINED syndrome, or Uncategorized error.

I_{FJCZP} *Error synchronization event* defines synchronized by Error synchronization events.

R_{JHNVT} If and only if all of the following are true, then on taking an error exception the PE error state is recorded as *Recoverable state (UER)*:

- The error has not been silently propagated by the PE.
- The error exception is not taken as a synchronous External abort to AArch32 state.
- The PE determines that software is able to recover execution from the preferred return address of the exception.
- One or more of the following are true:
  - The PE determines that software must take action to locate and repair the error to successfully recover execution. This might be because the exception was taken before the error was architecturally consumed by the PE, at the point when the PE was not be able to make correct progress without either consuming the error or otherwise making the state of the PE unrecoverable.
  - The implementation has elected to record the PE error state as Recoverable state (UER).
- The implementation has not elected to record the PE error state as Unrecoverable state (UEU), Uncontainable (UC), IMPLEMENTATION DEFINED syndrome, or Uncategorized error.

R_{MBVCF} If and only if all of the following are true, then on taking an error exception the PE error state is recorded as *Restartable state (UEO)*:

- The error has not been silently propagated by the PE.
- The error exception is not taken as a synchronous External abort to AArch32 state.
- The PE determines that software can recover execution from the preferred return address of the exception without the need for software to take action to locate and repair the error first.
- One or more of the following are true:
  - The error is an uncorrected error. This includes a deferred error.
  - The error is a corrected error and the error exception is not taken as an SError interrupt taken to AArch64 state.
  - The implementation has elected to record the PE error state as Restartable state (UEO).

- The implementation has not elected to record the PE error state as any of Recoverable state (UER), Unrecoverable state (UEU), Uncontainable (UC), IMPLEMENTATION DEFINED syndrome, or Uncategorized error.

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

- The error has been corrected and not silently propagated by the PE.
- The error exception is taken as an SError interrupt taken to AArch64 state.
- Software can recover execution from the preferred return address of the exception. Because the error has been corrected, software does not need to take action to locate and repair the error.
- The implementation has not elected to record the PE error state as any other type.

R_{NZYRP} If and only if all the following are true, then on taking an error exception the PE error state is recorded as an *Uncategorized error*:

- The error exception is taken as an asynchronous SError interrupt taken to AArch64 state.
- The implementation has elected to record the PE error state as an Uncategorized error.

R_{VHWHD} If and only if all the following are true, then on taking an error exception the PE error state is recorded as an IMPLEMENTATION DEFINED *syndrome*

- The error exception is taken as an asynchronous SError interrupt taken to AArch64 state.
- The implementation has elected to record the PE error state as an IMPLEMENTATION DEFINED syndrome.

I_{SRPJD} The IMPLEMENTATION DEFINED syndrome type might provide additional IMPLEMENTATION DEFINED syndrome recorded in the exception syndrome register. Software might be able to determine the state of the PE from this syndrome, or other IMPLEMENTATION DEFINED syndrome registers.

I_{WLZRP} Uncategorized error and IMPLEMENTATION DEFINED syndrome are defined for backwards compatibility with previous versions of the architecture. Arm does not recommend use of these PE error state values in new implementations that include other RAS features.

I_{VKMZB} The PE error states are summarized by Figure 2.1. Figure 2.1 assumes the type of error exception supports the resulting PE error state, never elects to record an error as a different PE error state when permitted, and does not show Uncategorized error or IMPLEMENTATION DEFINED syndrome.

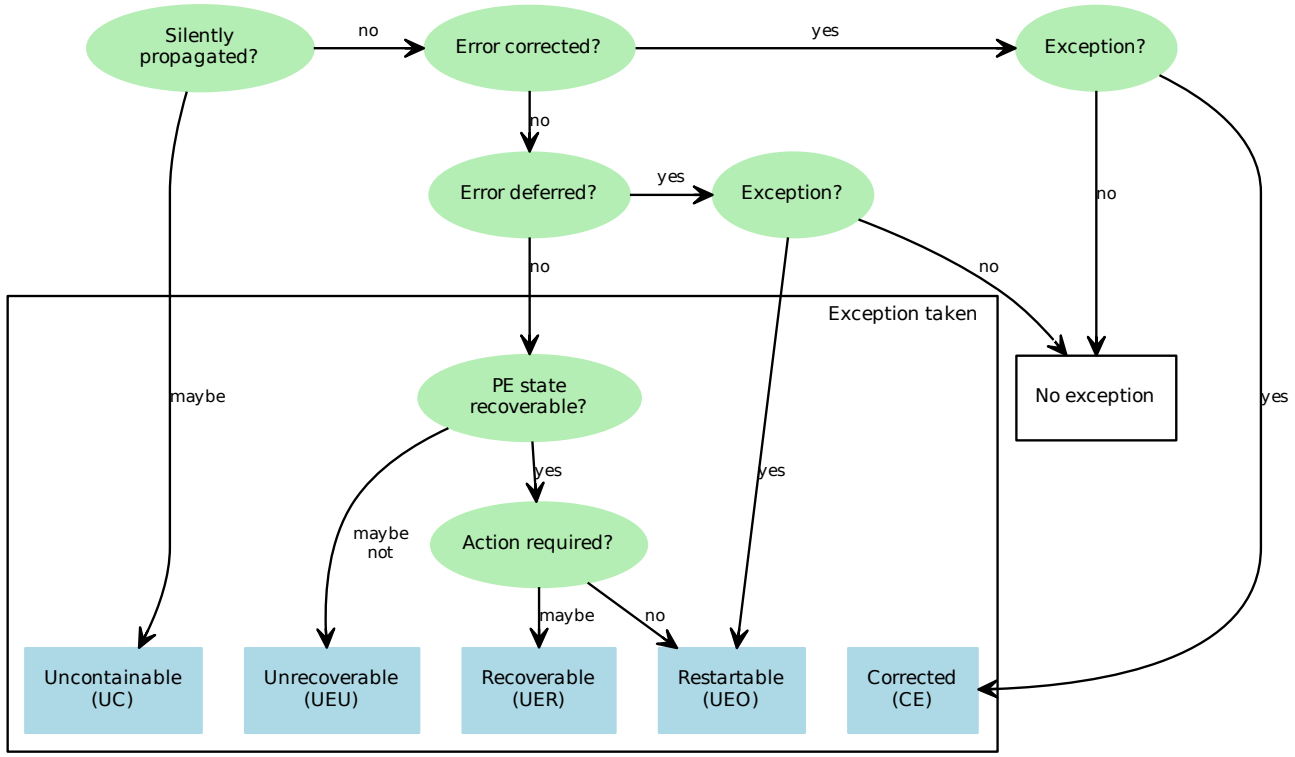

Figure 2.1: PE error states

IZQRGL If the PE error state reports that software can recover execution, or that software isolation might be possible because the error is synchronized by Error synchronization events, this does not necessarily mean that the error can be recovered from because the error in the system might be one which does not allow software to recover the operation. Rather, software *might* be able to recover if it can repair the error and continue.

For example, the component that originally detected the error and signaled it to the PE might record in a *RAS System Architecture* node that the error is uncontainable at the component, meaning the system has to be shut down to avoid catastrophic failure. The in-band error response to the PE is not required to signal the severity of the error to the PE. The recorded PE error state refers only to the PE, not the system error state.

If the in-band error response can signal the severity of the error to the PE, the PE might use this information to elect to report the PE error state as other than Recoverable state (UER). For example, if a processor cache detects an uncontainable tag RAM error, the PE might report the PE error state as Uncontainable (UC), even though the state of the PE itself is recoverable. However, this is not required, and software must not rely on this behavior and should determine from the system whether the error is recoverable at the system level.

See also:

- *PE error propagation*
- *Error synchronization event*

2.3.2.1 Using the PE error state classification

SXSKNS When the PE error state is recorded as Uncontainable (UC):

- Software must assume that either:
  - The error has been silently propagated by the PE.

  - Software is not able to recover execution from the preferred return address of the exception and the error was not synchronized by Error synchronization events.

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

S_{HYWFL} When the PE error state is recorded as Unrecoverable state (UEU):

- Software can assume the error has not been silently propagated by the PE.
- Software cannot safely recover execution from the preferred return address of the exception, even if it takes action to locate and repair the error. The state of the affected software, or both, is unrecoverable. However, if the software includes *Error synchronization event*s, software can use the properties of the Error synchronization event to determine which software is affected by the error.
- The affected software cannot continue and must be isolated by software.

S_{LSFYM} When the PE error state is recorded as Recoverable state (UER):

- The uncorrected error might remain latent in the system.
- If the exception handler takes action to locate and repair the uncorrected error, it can safely recover execution from the preferred return address of the exception. Otherwise on restart of the affected software the PE might attempt to consume the error again, causing a further error exception. If software cannot locate and repair the error, the affected software must be isolated by software.

S_{GLPZY} When the PE error state is recorded as Restartable state (UEO):

- The error might remain latent in the system.
- Software might take action to locate and repair the error before it is consumed. However, the affected software can be safely restarted by the exception handler without software taking any action to locate and repair the error.

For example, the error was signaled when the PE speculatively accessed corrupt data.

S_{GRZQS} When the PE error state is recorded as IMPLEMENTATION DEFINED syndrome or Uncategorized error, if software is not able to determine the actual state of the PE and memory, it should treat IMPLEMENTATION DEFINED syndrome and Uncategorized error as Uncontainable (UC).

2.3.3 Multiple SError interrupts

I_{CPJLW} Multiple physical and/or virtual SError interrupt conditions might be pending together. The architecture does not define relative priorities for asynchronous exceptions.

R_{DHKQZ} If multiple physical and/or virtual SError interrupt conditions are pending, it is IMPLEMENTATION DEFINED whether the multiple pending SError interrupt conditions are taken as a single SError interrupt exception.

R_{JBQSC} On taking an SError interrupt exception for more than one SError interrupt condition:

- If the exception is taken to AArch64 state and one or more pending SError interrupt conditions would be reported as IMPLEMENTATION DEFINED syndrome or Uncategorized error, then the syndrome recorded in ESR_ELx.ESS is IMPLEMENTATION DEFINED.
- Otherwise, the recorded PE error state applies recorded by combined effect of the errors.

I_{GNHXJ} Any pending SError interrupt conditions that are not taken with other SError interrupts as a single SError interrupt exception remains pending after the SError interrupt exception is taken.

2.3.4 Target Exception level for External abort and SError interrupt exceptions taken to AArch64 state

I_{NRZXZ} This section is included for completeness. It repeats the definitions from [1] and so is *non-normative*.

The default target Exception level for SError interrupt and synchronous External abort exceptions taken to AArch64 state is:

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

However:

- If EL3 is implemented and SCR_EL3.EA is `0b1`, all SError interrupt and synchronous external abort exceptions are taken to EL3.
- Otherwise, if EL2 is implemented and enabled in the current Security state, then:
  - If HCR_EL2.AMO is `0b1` or HCR_EL2.TGE is `0b1`, all SError interrupts from EL0 and EL1 are taken to EL2.
  - If HCR_EL2.TEA is `0b1` or HCR_EL2.TGE is `0b1`, all synchronous External abort exceptions from EL0 and EL1 are taken to EL2.

2.3.5 Target mode for External abort and SError interrupt exceptions taken to AArch32 state

I_{BMBXM} This section is included for completeness. It repeats the definitions from [1] and so is *non-normative*.

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

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

However:

- If EL3 is implemented and using AArch32 and SCR.EA is `0b1`:
  - All SError interrupt and synchronous external Data Abort exceptions are taken to Secure Monitor mode, using vector offset `0x10`.
  - All synchronous external Prefetch Abort exceptions are taken to Secure Monitor mode, using vector offset `0x0C`.
- Otherwise, if EL2 is implemented and using AArch32 and the PE is in Non-secure state:
  - If HCR.AMO is `0b1` or HCR.TGE is `0b1`, all SError interrupts from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.
  - If HCR.TEA is `0b1` or HCR.TGE is `0b1`, all synchronous External abort exceptions from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.

2.4 Error synchronization event

I_{YGRDP} The RAS Extension defines the Error synchronization event and the `ESB` instruction.

R_{GRJVN} An *Error synchronization event* is generated by any of the following:

- Executing an `ESB` instruction.
- When FEAT_IESB is implemented, and one of the following is true, taking an exception to an Exception level, ELx, using AArch64:
  - The appropriate SCTLR_ELx.IESB bit is `0b1`.
  - FEAT_DoubleFault is implemented, the Exception level is EL3, and SCTLR_EL3.NMEA is `0b1`.

  In Debug state this also applies to executing a `DCPSx` instruction to ELx.
- When FEAT_IESB is implemented, and one of the following is true, executing an exception return instruction at an Exception level, ELx, using AArch64:
  - The appropriate SCTLR_ELx.IESB bit is `0b1`.
  - FEAT_DoubleFault is implemented, the Exception level is EL3, and SCR_EL3.NMEA is `0b1`.

  In Debug state this also applies to executing a `DRET` instruction at ELx.

I_{PMGVM} In addition to generating an Error synchronization event, the `ESB` instruction might additionally record and then clear a masked pending asynchronous SError interrupt exception.

I_{NWLTG} For details of the operation and encoding of `ESB`, see [1].

I_{HQZRM} The FEAT_IESB feature and SCTLR_ELx.IESB bits are described by [1]. See also *Extension for synchronization at exception entry and return*.

I_{NNXVF} The FEAT_DoubleFault feature and SCR_EL3.NMEA bit are described by [1]. See also *Extension for synchronization at exception entry and return*.

R_{YZPBD} An error is *synchronized by Error synchronization events* if and only if all the following are true for each Error synchronization event:

- The error is generated by an instruction on the same PE as the Error synchronization event. This includes any memory accesses, instruction fetch, translation table walk, or hardware update to the translation tables made by the instruction.
- If the error exception for the error is taken in program order after the Error synchronization event completes, and either physical SError interrupt exceptions are unmasked when the Error synchronization event occurs or the error exception is taken synchronously, then all of the following are true:
  - The instruction that generated the error is in program order after the Error synchronization event.
  - On completion of the Error synchronization event, the PE state and memory system state are consistent with the PE having executed all instructions in program order before the Error synchronization event.
- If the error exception for the error is taken asynchronously as an SError interrupt, physical SError interrupt exceptions are masked when the Error synchronization event occurs, and the SError interrupt is not pending when the Error synchronization event completes, then all of the following are true:
  - The instruction that generated the error is in program order after the Error synchronization event.
  - On completion of the Error synchronization event, the PE state and memory system state are consistent with the PE having executed all instructions in program order before the Error synchronization event.

  The SError interrupt is not pending when the Error synchronization event completes if a subsequent read of ISR_EL1.A or ISR.A returns `0b0`.
- If the error exception for the error is taken asynchronously as an SError interrupt, the Error synchronization event is generated by an `ESB` instruction executed when physical SError interrupt exceptions are masked, and the `ESB` instruction does not set DISR_EL1.A or DISR.A to `0b1`, then all of the following are true:

  - The instruction that generated the error is in program order after the ESB.
  - On completion of the ESB, the PE state and memory system state are consistent with the PE having executed all instructions in program order before the ESB.

R_{NFKMQ} *Taken in program order after the Error synchronization event completes* means:

- For an Error synchronization event generated by an ESB instruction, the exception is taken in program order after the instruction.
- For an Error synchronization event generated by an exception return instruction when FEAT_IESB implemented, the exception is taken in program order after the instruction.
- For an Error synchronization event generated by an exception entry when FEAT_IESB is implemented, one of the following is true:
  - The exception is taken in program order strictly after the first instruction of the exception handler at the exception vector address.
  - The exception is taken from the first instruction of the exception handler at the exception vector address and the ESR_ELx.IESB syndrome bit is recorded as 0b0.

I_{QZSHG} The definition of synchronized by Error synchronization events means that if the error that is synchronized by Error synchronization events is generated by an instruction in program order before the Error synchronization event, then either the error exception is taken before the Error synchronization event, or on executing the Error synchronization event the following apply:

- If physical SError interrupt exceptions are unmasked or the error exception is taken synchronously, the Error synchronization event ensures that the error exception is not taken in program order after the Error synchronization event. This allows isolation of the software affected by the error.
- If physical SError interrupt exceptions are masked and the error exception is taken asynchronously:
  - If the Error synchronization event was generated by an ESB, the error is recorded in DISR_EL1 or DISR. Software can use the PE error state recorded in DISR_EL1 or DISR to determine what recovery is possible.
  - Otherwise, the error exception is pending when the Error synchronization event completes.

The SError interrupt might have been pending before or made pending by the Error synchronization event.

The definition does not mean that if the error is generated by a instruction in program order after the Error synchronization event, then the error exception will only be taken after the Error synchronization event. The error exception might be taken before the Error synchronization event, if the PE speculated past the Error synchronization event and speculatively executed the instruction that generated the error. This might cause software to generate a false failure. Error synchronization events are not speculation barriers.

I_{SQCFG} It is implementation-specific which physical errors are synchronized by Error synchronization events. However, the criteria for the PE error state mean that if the PE reports the PE error state as one of the following, the error must be either explicitly or implicitly synchronized by Error synchronization events:

- Unrecoverable state (UEU).
- Recoverable state (UER).
- Restartable state (UEO).

This is because *synchronized by Error synchronization events* is a criterion for Unrecoverable state (UEU), and the criteria for Recoverable state (UER) and Restartable state (UEO) satisfy the definition of synchronized by Error synchronization events.

For other physical errors:

- An error that has been silently propagated by the PE and is not reported as either IMPLEMENTATION DEFINED syndrome or Uncategorized error must be reported as Uncontainable (UC) and is not containable even if synchronized by Error synchronization events. Software must assume the error has been silently propagated even if the error is synchronized by Error synchronization events.

- It is implementation-specific whether an error reported with an ESR_ELx.ESS syndrome that is IMPLEMENTATION DEFINED syndrome or Uncategorized error is synchronized by Error synchronization events.
- The following errors have not been consumed by the PE:
  - A *Deferred error*.
  - A *Corrected error*.
  - An error exception from a read by hardware speculation that does not corrupt the state of the PE.

  Software can recover execution from these errors regardless of whether the error is synchronized by Error synchronization events.
- An implementation might have other IMPLEMENTATION DEFINED sources of SError interrupt, see R_{FNVVJ}. If an IMPLEMENTATION DEFINED SError interrupt is generated by a level-sensitive interrupt signal, it cannot be synchronized by Error synchronization events.

I_{VFFYW} An Error synchronization event might operate as follows:

(1) The PE ensures that any error synchronized by Error synchronization events and generated by an instruction in program order before the Error synchronization event has caused a physical SError interrupt exception to become pending.

(2) If a physical SError interrupt is pending for an error synchronized by Error synchronization events and generated by an instruction in program order before the Error synchronization event, and physical SError interrupt exceptions are not masked at the current Exception level, then the physical SError interrupt exception is taken before completion of the Error synchronization event. The SError interrupt might have been made pending by the Error synchronization event, or might have been pending before the Error synchronization event.

I_{RDWTF} The prioritization of asynchronous interrupts is IMPLEMENTATION DEFINED. This means the PE might take another exception before an SError interrupt made pending by the Error synchronization event. In this case, the SError interrupt remains pending.

Arm recommends the SError interrupt is prioritized over other exceptions.

R_{NPPGJ} If an SError interrupt for an error synchronized by Error synchronization events is pending after completing the Error synchronization event generated by an `ESB` instruction, and physical SError interrupt exceptions are masked at the current Exception level, the `ESB` instruction performs the following steps:

(1) The pending physical SError interrupt is recorded in DISR_EL1 or DISR. This includes the PE error state that the pending error exception would record if taken.

(2) The DISR_EL1.A bit or DISR.A bit is set to `0b1`.

(3) The pending state of the physical SError interrupt is cleared.

The SError interrupt might have been made pending by the Error synchronization event, or might have been pending before the Error synchronization event.

R_{BLRTM} The criteria for `ESB` recording the PE error state in DISR_EL1 or DISR are the same as for that for recording the PE error state in ESR_ELx or DFSR when an SError interrupt exception taken to the current execution state.

R_{KNWBN} If an SError interrupt is taken as part of an Error synchronization event generated by an `ESB` instruction, the `ESB` instruction address is the *preferred return address* of the exception.

Note

See [1] for the definition of the *preferred return address* for an exception.

R_{SFHDS} On executing an `ESB` instruction when SError interrupt exceptions are masked, any pending SError interrupt generated by an error that is not synchronized by Error synchronization events:

- Remains pending after completion of the Error synchronization event.
- Does not update DISR_EL1 or DISR.

I_{CQKXL} The error recovery, fault handling, and critical error interrupts described by *RAS System Architecture* are asynchronous interrupts, not errors, and so are not synchronized by Error synchronization events.

I_{BBGXN} If multiple SError interrupt conditions are pending, an Error synchronization event synchronizes all errors that are synchronized by Error synchronization events.

S_{VFHGT} Software must be aware that an SError interrupt taken at an Error synchronization event or recorded in the DISR_EL1 or DISR register by an `ESB` instruction might have been generated by hardware speculation of an instruction in program order after the Error synchronization event.

2.4.1 `ESB` and Virtual SError interrupt exceptions

R_{LLLVR} If all of the following are true, then an `ESB` instruction executed at EL0 or EL1 synchronizes a pending virtual SError interrupt:

- EL2 is implemented and enabled in the current Security state.
- Any of the following are true:
  - EL2 is using AArch64, HCR_EL2.AMO is `0b1`, HCR_EL2.TGE is `0b0`, and HCR_EL2.VSE is `0b1`.
  - EL2 is using AArch32, HCR.AMO is `0b1`, HCR.TGE is `0b0`, and HCR.VA is `0b1`.
- The VSESR_EL2 and, if implemented, VDFSR registers are writable. See *Fields in VSESR_EL2, VDFSR, DISR(_EL1), and VDISR(_EL2)*.

In these cases, a virtual SError interrupt is pending, and the following occur when an `ESB` instruction is executed at EL0 or EL1:

- If the virtual SError interrupt is unmasked at the current Exception level, it is taken before the completion of the `ESB` instruction.
- If the virtual SError interrupt is masked at the current Exception level:
  - HCR_EL2.VSE or HCR.VA cleared to `0b0`.
  - The virtual SError interrupt syndrome from VSESR_EL2 or VDFSR is recorded in VDISR_EL2 or VDISR. See R_{HDCTW} and R_{FLYGZ}.
  - VDISR_EL2.A or VDISR.A is set to `0b1` to indicate the SError interrupt was pending prior to the execution of the `ESB` instruction.

R_{GXHYX} If all of the following are true, then it is IMPLEMENTATION DEFINED whether or not an `ESB` instruction executed at EL0 or EL1 synchronizes a pending virtual SError interrupt:

- EL2 is implemented and enabled in the current Security state.
- Any of the following are true:
  - EL2 is using AArch64, HCR_EL2.AMO is `0b1`, HCR_EL2.TGE is `0b0`, and HCR_EL2.VSE is `0b1`.
  - EL2 is using AArch32, HCR.AMO is `0b1`, HCR.TGE is `0b0`, and HCR.VA is `0b1`.
- The VSESR_EL2 and, if implemented, VDFSR registers are implemented as RAZ/WI. See *Fields in VSESR_EL2, VDFSR, DISR(_EL1), and VDISR(_EL2)*.

In these cases, a virtual SError interrupt is pending, If the `ESB` instruction synchronizes a pending virtual SError interrupt in this case, then the following occur when an `ESB` instruction is executed at EL0 or EL1:

- If the virtual SError interrupt is unmasked at the current Exception level, it is taken before the completion of the `ESB` instruction.
- If the virtual SError interrupt is masked at the current Exception level:
  - HCR_EL2.VSE or HCR.VA cleared to `0b0`.

- The virtual SError interrupt syndrome in VDISR_EL2 or VDISR is set to zero. See R_{HDCTW} and R_{FLYGZ}.
- VDISR_EL2.A or VDISR.A is set to 0b1 to indicate the SError interrupt was pending prior to the execution of the ESB instruction.

If the ESB instruction does not synchronize a pending virtual SError interrupt, then an ESB instruction executed at EL0 or EL1 ignores the pending virtual SError interrupt and the virtual SError interrupt stays pending.

R_{YVBSH} If all of the following are true, then it is IMPLEMENTATION DEFINED whether or not an ESB instruction executed at EL0 or EL1 synchronizes a pending virtual SError interrupt from an IMPLEMENTATION DEFINED source:

- EL2 is implemented and enabled in the current Security state.
- Any of the following are true:
  - EL2 is using AArch64, HCR_EL2.AMO is 0b1, and HCR_EL2.TGE is 0b0.
  - EL2 is using AArch32, HCR.AMO is 0b1, and HCR.TGE is 0b0.

If a virtual SError interrupt from an IMPLEMENTATION DEFINED source that is synchronized by Error synchronization events is pending, then the following occur when an ESB instruction is executed at EL0 or EL1:

- If the virtual SError interrupt is unmasked at the current Exception level, it is taken before the completion of the ESB instruction.
- If the virtual SError interrupt is masked at the current Exception level:
  - The pending state of the virtual SError interrupt is cleared.
  - The virtual SError interrupt syndrome is set to the IMPLEMENTATION DEFINED syndrome for the virtual SError interrupt. See R_{YZCYX} and R_{JQGXD}.
  - VDISR_EL2.A or VDISR.A is set to 0b1 to indicate the SError interrupt was pending prior to the execution of the ESB instruction.

If a virtual SError interrupt from an IMPLEMENTATION DEFINED source that is not synchronized by Error synchronization events is pending, then an ESB instruction executed at EL0 or EL1 ignores the pending virtual SError interrupt and the virtual SError interrupt stays pending.

Note

R_{LLLVR}, R_{GXHYX}, and R_{YVBSH} happen in parallel with the Error synchronization event for physical SError interrupt exceptions.

2.4.2 Extension for synchronization at exception entry and return

I_{DYZRN} The FEAT_IESB feature adds a control bit to each AArch64 SCTLR_ELx System register to insert an implicit Error synchronization event at exception entry and exception return. For the register field descriptions, see [1].

R_{DPSJR} The rules in this section apply when FEAT_IESB is implemented.

I_{WDSBL} An implicit Error synchronization event has no effect on DISR_EL1 or VDISR_EL2.

R_{KJWNS} When FEAT_DoubleFault is implemented, and the Effective value of SCR_EL3.NMEA is 0b1, SCTLR_EL3.IESB is ignored and its Effective value is 0b1.

2.4.2.1 Synchronization on exception entry

R_{RNZWY} For each value of ELx in EL1, EL2, EL3, if all of the following are true, then each exception that is taken to ELx generates an Error synchronization event:

- ELx is using AArch64.
- The Effective value of SCTLR_ELx.IESB is 0b1.

R_{RPBWR} For each value of ELx in EL1, EL2, EL3, if all of the following are true, then executing a DCPSx instruction generates an Error synchronization event:

- The PE is in Debug state.
- ELx is using AArch64.
- The Effective value of SCTLR_ELx.IESB is 0b1.

R_{JQSKQ} If an SError interrupt exception is taken to the Exception level ELy as a result of the Error synchronization event generated on exception entry by the FEAT_IESB mechanism, then all the following occur:

- The PE sets the ESR_ELy.IESB bit in the SError interrupt exception syndrome to 0b1.
- The preferred return address for the SError interrupt exception is the exception vector address for the original exception.

Note

ELy might be the same Exception level as ELx.

I_{FWZHV} If SError interrupt exceptions are masked at ELx, any SError interrupt made pending by the Error synchronization event stays pending.

U_{MMVJW} The prioritization of asynchronous interrupts is IMPLEMENTATION DEFINED. This means that an implementation might choose to behave as if the SError interrupt was taken before the implicit Error synchronization event, if the SError interrupt was not masked, taking the SError interrupt in place of the exception.

In this case, ESR_ELy.IESB is set to 0b0 and the reported PE error state correctly indicates, for instance, whether software can recover execution from the preferred return address for the SError interrupt in ELR_ELy.

When FEAT_DoubleFault is implemented, Arm recommends that the implicit Error synchronization event is inserted before taking an exception to EL3.

2.4.2.2 Synchronization on exception return

R_{SKRCR} For each value of ELx in EL1, EL2, EL3, if all of the following are true, then executing an exception return instruction at ELx generates an Error synchronization event:

- The instruction does not generate any exception.
- ELx is using AArch64.
- The Effective value of SCTLR_ELx.IESB is 0b1.

Note

On an illegal return event the exception return instruction sets PSTATE.IL to 0b1, which causes the next instruction to generate an Illegal State exception. The exception return instruction does not generate the exception.

R_{CVPDN} For each value of ELx in EL1, EL2, EL3, if all of the following are true, then executing an DRPS instruction at ELx generates an Error synchronization event:

- The PE is in Debug state and the instruction does not generate any exception.
- ELx is using AArch64.
- The Effective value of SCTLR_ELx.IESB is 0b1.

R_{GXQYD} Any SError interrupt exception taken as part of the Error synchronization event terminates execution of the instruction.

R_{LPKVM} If an SError interrupt exception is taken to an Exception level, ELy, as a result of the Error synchronization event generated on exception return by the FEAT_IESB mechanism, then all the following occur:

- The PE sets the ESR_ELy.IESB bit in the SError interrupt exception syndrome to an IMPLEMENTATION DEFINED choice of `0b0` or `0b1`.
- The preferred return address for the SError interrupt is the address of the `ERET` instruction.

I_{JZHDB} If SError interrupt exceptions are masked at ELx, any SError interrupt made pending by the Error synchronization event stays pending.

2.4.3 Error synchronization barriers in a minimal implementation

I_{GQQCK} Error synchronization events and the `ESB` instruction can be implemented as no-ops if all of the following apply:

- Either there are no sources of SError interrupts, or all SError interrupts are reported as Uncategorized error and not synchronized by Error synchronization events.
- Either EL2 is not implemented, or VSESR_EL2 and VDFSR are implemented as RAZ/WI. See *Fields in VSESR_EL2, VDFSR, DISR(_EL1), and VDISR(_EL2)*.

This allows for a very low cost implementation of the RAS Extension.

2.5 Virtual SError interrupts

I_{LSSCN} When implemented, EL2 provides a virtual SError interrupt.

Virtual SError interrupts are generated by one of the following:

- Software sets HCR_EL2.AMO to `0b1` to enable the virtual SError interrupt mechanism and HCR_EL2.VSE to `0b1` to inject a virtual SError interrupt. In AArch32 state these are the HCR.AMO and HCR.VA bits respectively.
- An IMPLEMENTATION DEFINED source of virtual SError interrupts.

The RAS Extension provides:

- Mechanisms to allow a hypervisor to specify the syndrome value reported to a guest Operating System on taking a virtual SError interrupt injected using HCR_EL2.VSE or HCR.VA.
- Support for EL0 or EL1 to isolate a virtual SError interrupt injected using the HCR_EL2.VSE or HCR.VA mechanism as if it were a physical SError interrupt. See *ESB and Virtual SError interrupt exceptions*.

When the RAS Extension is implemented:

R_{HDCTW}
- When a virtual SError interrupt injected using HCR_EL2.VSE is taken to EL1 using AArch64, the PE sets ESR_EL1.ESS to the value of the *Virtual syndrome register*, VSESR_EL2.

R_{FLYGZ}
- When a virtual SError interrupt injected using HCR_EL2.VSE or HCR.VA is taken to EL1 using AArch32, DFSR.{AET,ExT} are set to values from VSESR_EL2 or VDFSR.

  The remainder of DFSR is set as defined by VMSAv8-32.

R_{YZCYX}
- When a virtual SError interrupt from an IMPLEMENTATION DEFINED source is taken to EL1 using AArch64, ESR_EL1.ESS is set to an IMPLEMENTATION DEFINED value that must report the PE error state as either:
  - An IMPLEMENTATION DEFINED syndrome. That is, ESR_EL1.ESS[24] is `0b1`.
  - An Uncategorized error. That is, ESR_EL1.ESS is zero.

R_{JQGXD}
- When a virtual SError interrupt from an IMPLEMENTATION DEFINED source is taken to EL1 using AArch32, DFSR.{AET,ExT} are set to IMPLEMENTATION DEFINED values.

Note

See [1] for descriptions of VSESR_EL2 and VDFSR.

2.6 Error records in the PE

I_{KMMPK} A component that records detected errors is called a node by the *RAS System Architecture*. Each node implements one or more error records.

R_{VNLPC} It is IMPLEMENTATION DEFINED whether the processor that implements a PE implements any nodes.

R_{XKDRX} A PE implementing the RAS Extension might implement the System register interface to nodes.

I_{SCVSB} The System register interface to nodes is not restricted to accessing only PE nodes.

I_{ZRKKQ} When an error is recorded by a PE node, one or more of the following might be generated, according to the configuration of the node:

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

See also:

- *Error record System register view*
- *RAS System Architecture*
- *Nodes*

2.6.1 Error record System register view

I_{SLVDW} If the System register interface to a node is implemented, software accesses the error records of the node using Error record System registers.

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

I_{NWBNQ} The AArch64 Error record System registers are those registers with an ERX*_EL1 mnemonic. See *Using AArch64 System registers*.

The AArch32 Error record System registers are those registers with an ERX* mnemonic. See *Using AArch32 System registers*.

These registers are defined in [1].

I_{VVMCQ} The error record register contents are described by *Error record registers, including memory mapped view*.

R_{ZBCFZ} If FEAT_RASv1p1 is implemented, all error records accessible through System registers implement RAS System Architecture v1.1.

S_{VBBNY} To access an error record, software:

1. Sets the error selection register, ERRSELR_EL1.SEL or ERRSELR.SEL, to the index of the record being accessed.
2. Accesses the error record using the ERX*_EL1 or ERX* System registers.

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

See also:

- *Synchronization and error record accesses*
- *Error record registers, including memory mapped view*

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

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

For a given implementation:

- If ESB never synchronizes any errors, then DISR_EL1.A and DISR.A might be RES0.
- The error syndrome registers are capable of storing any syndrome value that might be reported by hardware on taking a physical error exception.
- If any of ESR_ELx[24:0], HSR[11:9], and DFSR[15:14,12] is not used and always set to zero by hardware on taking a physical SError interrupt exception or synchronous External Abort exception, it can be RES0 in that syndrome register.
- A bit that is not used and always set to zero or always set to one by hardware on taking a physical SError interrupt is permitted to be RES0 or RES1 respectively in the corresponding other syndrome registers. See Table 2.2.

In Table 2.2, the bit described in the left-hand column is permitted to be RES0 or RES1 if the corresponding bit is always set to zero or always set to one (respectively) on taking an SError interrupt in all of the registers listed in the other columns marked *Yes* on that row.

Table 2.2: Permitted relaxations for bits in error syndrome registers

Bit that is permitted to be RES0 or RES1	**ESR_ELx[x], $x \in$ [24:0]**	**HSR[x], $x \in$ [11:9]**	**DFSR[x], $x \in$ [15:14,12]**
VSESR_EL2[x]	Yes	-	Yes
VDISR_EL2[x]	Yes	-	Yes
DISR_EL1[x]	Yes	-	-
VDFSR[x]	-	-	Yes
VDISR[x]	-	-	Yes
DISR[x]	-	Yes	Yes

Note

I_{SLNMV} means that VSESR_EL2, VDFSR, DISR_EL1, DISR, VDISR_EL2, and VDISR can be implemented as RAZ/WI when all of the following apply:

- ESB never synchronizes any errors, including virtual System errors. That is, ESB executes as a no-op.
- ESR_ELx[24:0], HSR[11:9], and DFSR[15:14,12] are always set to zero by hardware on taking a physical SError interrupt exception or synchronous External Abort exception. This means that the PE error state is always reported as Uncategorized error when a physical SError interrupt is taken to AArch64 state.

This allows for a very low cost implementation of the RAS Extension.

Chapter 3
RAS System Architecture

I_{XKHGG} The *Reliability, Availability, Serviceability* (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 *Processing element* (PE).

R_{DKJPB} A *node* is a RAS System Architecture component that records an error detected or consumed by a system component.

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 components that implement the RAS System Architecture.

3.1 Nodes

R_{RDHHP} A component might implement one or more nodes.

The RAS System Architecture defines the following common features for a node:

R_{XMFKF} **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 the node fully disables error detection and correction when reporting and recording are disabled.

See *Detecting and consuming errors*.

R_{FNBYQ} ***Fault handling interrupt***
Asynchronous reporting of all or some recorded errors by an interrupt, that is, Corrected errors, Deferred errors, and Uncorrected errors. It is IMPLEMENTATION DEFINED whether a node provides a single control for all errors, or a first control for Corrected errors and a second control for all other detected errors.

See *Fault handling interrupt*.

R_{QQRSQ} ***Corrected error counter***
It is IMPLEMENTATION DEFINED whether a node implements a counter for counting Corrected errors. Software might 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*.

R_{WFWCL} **Timestamps**
It is IMPLEMENTATION DEFINED whether a node records a timestamp in each error record.

See *Timestamp extension*.

R_{ZMMBH} ***In-band error response*** **(external abort)**
In-band signaling of detected Uncorrected errors to the consumer of the error. It is also referred to as an external abort. Corrected errors and Deferred errors are not reported by such means.

See *In-band error response signaling (external aborts)*.

R_{VHDZW} ***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, Deferred errors are not reported by this means.

See *Error recovery interrupt*.

R_{BJNDJ} ***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*.

R_{RFNHX} **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*.

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, each error record has the same features and all error records share the controls.

Note

If a component requires multiple sets of controls, 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.

3.1.1 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.

U_{GMKHP} Using a single error record is efficient for the implementation.

However, consider an example node for an SoC memory controller component that records errors detected within both:

- An internal buffer that acts as a queue for memory accesses.
- An external memory module, that is, an external *Field Replaceable Unit* (FRU).

In this node, using a single error record for errors from either source might lead to the following scenarios:

(1)
- A Corrected error is detected in the internal buffer and recorded in the error record.
- Before software processes the error record, an Uncorrected error is detected in the external FRU.

(2)
- A Corrected error is detected in the external FRU and recorded in the error record.
- Before software processes the error record, an Uncorrected error is detected in the external FRU.

In both scenarios, the second error overwrites the syndrome for the first error, because *Writing the error record* requires this. It is IMPLEMENTATION DEFINED what information, if any, is retained for the first error in the IMPLEMENTATION DEFINED parts of the syndrome.

This means the two scenarios might be indistinguishable to software. In particular any indication of where the Corrected error was detected in the syndrome for the first error might be overwritten by the second error. If this is the case, software will need to treat the two scenarios identically, that is, as if there was a corrected internal error.

However, an internal error might be considered more significant than an external FRU error. For example, because the external FRU is *field-replaceable* whereas the SoC is not. Implementing separate error records for the internal buffer and external FRU would avoid this issue.

Implementations should therefore consider the impact such choices might have on the serviceability and availability of the system.

If a single node implements multiple error records, then all of the following are true:

R_{VRMSL}
- The error records are indexed sequentially within a group of error records starting from the first error record for the node.

R_{HCXWW}
- 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`.
  - ERR<n>FR[63:2] are RES0.
  - The ERR<n>CTLR register is RES0.

R_{RFPVW} A *group of error records* consists of the error records of one or more nodes.

R_{DBPFH} A group of error records 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*.

I_{TPDYQ} An example of a group of error records containing five error records owned by three nodes might be arranged as shown in Figure 3.1:

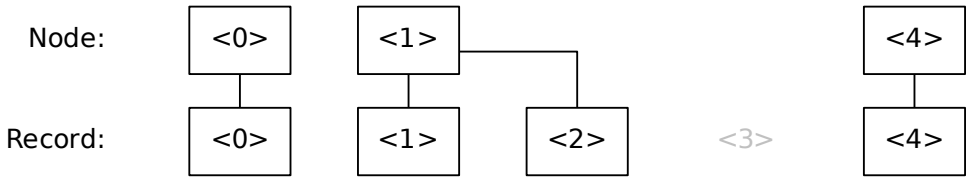

Figure 3.1: A group containing five error records owned by three nodes

- Node <0> owns a single error record: <0>. ERR0FR describes the features for this node, and ERR0CTLR contains the controls for this node.
- Node <1> owns two error records: <1> and <2>.
  - ERR1FR describes the features for this node, and ERR1CTLR contains the controls for this node.
  - ERR2FR.ED is `0b00` and ERR2CTLR is not implemented.
- Error record <3> is not implemented. ERR3FR.ED is `0b00`, and ERR3CTLR, ERR3STATUS, ERR3ADDR, and ERR3MISC<m> are not implemented.
- Node <4> owns a single error record: <4>. ERR4FR describes the features for this node, and ERR4CTLR contains the controls for this node.
- If the group of error records is accessed using a memory-mapped view then ERRDEVID.NUM is 5.
- If the group of error records is accessed using System registers then ERRIDR_EL1.NUM is 5.

3.1.2 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.

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.

For example, if 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, then the cache should treat it 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.

It is IMPLEMENTATION DEFINED whether:

R_{VYRXT}
- A Requester that consumes a signaled detected error reports the consumed error.

R_{LRQSG}
- Errors are reported when a detected error is propagated between components.

R_{WDJGD}
- All corrected errors are reported.

R_{GVPMK}
- 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.

When an error is detected or consumed by a component:

R_{LBHMF} • If the error can be corrected:
  – The error is corrected.
  – Optionally, the detected error is reported to a node, the node records a Corrected error, and if implemented and enabled, a fault handling interrupt is raised.
  – If the error is detected on a read access by a consumer, corrected data is returned to the consumer.

R_{LMCVC} • 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 consumer.
  – The error is reported to a node and the node records a Deferred error.
  – 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} • If the error cannot be corrected and cannot be deferred:
  – The error is reported to a node and the node records an Uncorrected error.
  – 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 consumer, and if implemented and enabled, a in-band error response is returned to the consumer.
  – 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 consumer that is incapable of receiving a deferred error response, then it is not possible to defer the error to the consumer.

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 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.

See also:

- *Standard error record*
- *Error recovery interrupt*
- *Fault handling interrupt*
- *In-band error response signaling (external aborts)*
- *Standard format Corrected error counter*

3.2 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.

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} If RAS System Architecture v1.0 is implemented, there are two ERR<n>MISC<m> for each error record:

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

R_{QCKVG} If RAS System Architecture v1.1 is implemented, there are 4 ERR<n>MISC<m> for each error record:

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

Note

The RAS System Architecture permits the implementation of ERR<n>MISC2 and ERR<n>MISC3 in implementations of the RAS System Architecture v1.0.

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

I_{PVYZG} *Error record System register view* defines System registers for accessing a group of error records.

I_{PBJTL} *Memory-mapped view* defines reusable formats for a memory-mapped views of error records. Use of reusable formats by any component in the system is OPTIONAL.

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

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

3.2.1 Component error states

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

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

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

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 error* (CE):

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

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 error* (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 error.
- 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 as service failure mode and continues to operate.
- The implementation has not elected to record the component error state as Uncorrected error.

Note

A Deferred error might be recorded for an error that cannot be corrected. However, for the purposes of the component error state taxonomy, Deferred error is classified separately from Uncorrected error.

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 error*:

- At least one of 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 error.
- The error is latent in the system.

R_{WHGSP} An Uncorrected error is recorded as one of the following sub-types:

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

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

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

If the error cannot be isolated, 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 a Uncorrected error, the component error state is recorded as *Unrecoverable error* (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 error as Unrecoverable error.
- The implementation has not elected to record the error as Uncontainable error.

R_{CNBRY} If and only if all of the following are true, then on recording a Uncorrected error, the component error state 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 was not recorded as a Deferred error.
- The implementation has not elected to record the error as Unrecoverable error, or Uncontainable error.

R_{FFTXZ} If and only if all of the following are true, then on recording a Uncorrected error, the component error state 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 error as Deferred error, Unrecoverable error, or Uncontainable error.

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

Note

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} If and only if all of the following are true, then on recording a Uncorrected error, the component error state is recorded as *Recoverable error* (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 error as Recoverable error.
- The implementation has not elected to record the error as Deferred error, Unrecoverable error, or Uncontainable error.

R_{CFZTH} If and only if all of the following are true, then on recording a Uncorrected error, the component error state is recorded as *Restartable error* (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 error as Deferred error, Unrecoverable error, or Uncontainable error.

I_{TVJNM} The component error state types are summarized by Figure 3.2. Figure 3.2 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.

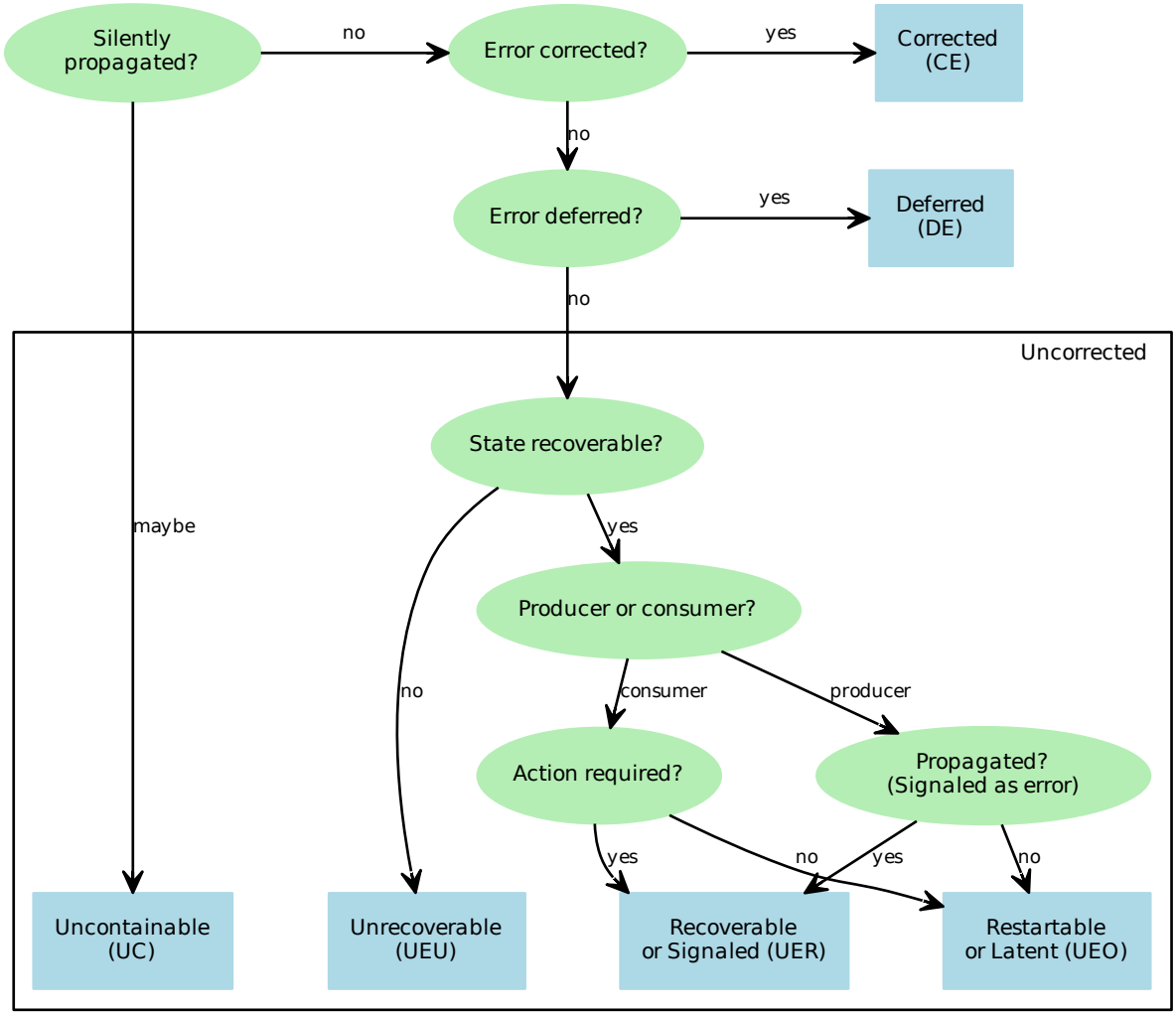

Figure 3.2: Component error state types

3.2.2 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.
- 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_{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.

3.2.2.1 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 3.1.

In Table 3.1, **V**, **CE**, **DE**, **UE**, **UET** refer to fields in ERR<n>STATUS.

Table 3.1: Encoding the highest priority component error state

V	CE	DE	UE	UET	Highest priority component error state type	Mnemonic
0	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	None	-
1	0b00	0	0	UNKNOWN	None	-
1	!= 0b00	0	0	UNKNOWN	Corrected error	CE
1	X	1	0	UNKNOWN	Deferred error	DE
1	X	X	1	0b10	Uncorrected error: Latent error or Restartable error	UEO
1	X	X	1	0b11	Uncorrected error: Signaled error or Recoverable error	UER
1	X	X	1	0b01	Uncorrected error: Unrecoverable error	UEU
1	X	X	1	0b00	Uncorrected error: Uncontainable error	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.

3.2.2.2 Prioritizing errors, RAS System Architecture v1.0

R_{ZPTXT} When RAS System Architecture v1.0 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 3.2.

In Table 3.2:

- 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 3.1, 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, 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:

- `0`: Count the error if a corrected error counter is implemented. Keep the previous error syndrome.
- `1`: Count the error. If ERR<n>STATUS.OF == 1 before the error is counted, 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, 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, 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 3.2: RAS System Architecture v1.0 rules for overwriting error records

	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

3.2.2.3 Prioritizing errors, RAS System Architecture v1.1

R_{PNFPB} When RAS System Architecture v1.1 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 3.3.

In Table 3.3:

- 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 3.1, 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, set ERR<n>STATUS.OF to `0b1`.

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

- `0`: The behavior is the same as **CO**.
- `1`: Count the error. If ERR<n>STATUS.OF == 1 before the error is counted, 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, set ERR<n>STATUS.OF to `0b1`.

Table 3.3: RAS System Architecture v1.1 rules for overwriting error records

	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

3.2.2.4 Overwriting the error syndrome

R_{RVGRM} When the node records an error in an error record and the previous syndrome is *overwritten* with the new error syndrome:

- Modifies ERR<n>STATUS.{V, CE, DE, UE} to indicate the new component error state, as described by Table 3.1:
  - Fields shown as `X` in Table 3.1 are unchanged.
  - Other ERR<n>STATUS.{V, CE, DE, UE} fields are set to the value given in Table 3.1.

  If the component error state is Corrected error, 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 error is a type of Uncorrected error, 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, 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, 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, it 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, 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 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.

3.2.2.5 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 error, then ERR<n>STATUS.UE is set to `0b1`.
  - If Deferred error, then ERR<n>STATUS.DE is set to `0b1`.
  - If Corrected error, 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 error is a type of Uncorrected error.
- ERR<n>STATUS.{ER, PN, IERR, SERR}, ERR<n>ADDR, and ERR<n>STATUS.AV are unchanged.
- If the RAS Timestamp Extension is implemented, 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`.

3.2.2.6 Detecting multiple errors

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

R_{ZJXMD} If a corrected error counter is implemented, and multiple countable errors are detected simultaneously, it is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether all the errors are counted.

3.2.3 Error syndrome

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

3.2.3.1 Corrected error field

I_{BRMPK} When the syndrome for a Corrected error is recorded, the node can indicate through the ERR<n>STATUS.CE error type field one of the following:

- 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.

3.2.3.2 Poison indicator

I_{DTYHM} If supported by a node, when the syndrome for a Deferred error or Uncorrected error 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 error as an Uncorrected error. For example, because the component does one or more of:
    - Enters a service failure mode.
    - Signals an in-band error response to the consumer of the data.
    - Propagates the value to a component that does not support poison. This is an Uncontainable error.
  - If the component has deferred the error again, records the error as a Deferred error. 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 this as a Deferred error with ERR<n>STATUS.PN set to `0b0`.

Therefore when software examines the error records, a 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. For example, a SECDED scheme where the poisoned value has a Hamming distance greater than 2 bits from any valid 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`.

3.2.4 Security and Virtualization

I_{BQFLD} Access to the Error System register view of error record registers can be controlled using Trap exceptions. See [1].

R_{VXDJW} If a PE implements System register access to error records for a component that processes Secure data, then either:

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

I_{JTWWK} The definition of *Secure data* is implementation-specific and depends on how the information encoded in the data relates to the threat model for the system.

For example, in a typical system that supports both Secure and Non-secure memory, data stored in or related to Secure memory is considered Secure data, and other data is considered *Non-secure data*.

R_{NLZMH} If a memory-mapped component processes Secure data, then one of the following applies:

- The error records are visible only to Secure accesses.
- The error records have reduced visibility to Non-secure accesses, that does not affect operation in Secure state, does not provide visibility of Secure data, or both.

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

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

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

3.2.5 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.

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

R_{WFPWF} Direct reads and writes of the error records through the ERX*_EL1 AArch64 System registers are indirect reads of ERRSELR_EL1.

R_{FZBLM} Direct reads and writes of the error records through the ERX* AArch32 System registers are indirect reads of ERRSELR.

3.2.6 Bridges to other architectures

R_{LWGCK} A *bridge* is a component that passes transactions between two domains.

I_{LHGZM} For example, 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.

U_{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 domains.

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*.

See also:

- *Multiple error records per node*

3.2.7 Software faults

I_{SSQXP} Examples of *software faults* include:

- Access to memory or peripheral register that is not present. This includes cases where Secure and Non-secure memory 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.
  - [1] 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, .
  - [1] 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, 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
13	Illegal address (software fault). For example, access to unpopulated memory.
14	Illegal access (software fault). For example, byte write to word register.
15	Illegal state (software fault). For example, device not ready.
25	Error recorded by PCIe error logs. Indicates that the node 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.

3.2.8 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.

3.3 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.

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.

3.4 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.

R_{WHXMB} For a node <n>, if fault handling interrupt is implemented, then the control for generating the fault handling interrupt on corrected error events, ERR<n>CTLR.CFI, might be implemented.

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_{TSPRZ} For a node <n>, if the ERR<n>CTLR.CFI control is implemented, then the ERR<n>CTLR.FI control is implemented, and the fault handling interrupt is *enabled* for Deferred errors and Uncorrected errors when ERR<n>CTLR.FI is `0b1` and *disabled* for Deferred errors and Uncorrected errors when ERR<n>CTLR.FI is `0b0`.

R_{TSXMY} For a node <n>, if the ERR<n>CTLR.CFI control is not implemented, then the control for generating the fault handling interrupt on all recorded errors, ERR<n>CTLR.FI, might be implemented.

R_{ZCKJL} For a node <n>, if the ERR<n>CTLR.FI control is implemented and the ERR<n>CTLR.CFI control is not implemented, then the fault handling interrupt is *enabled* for corrected error events, Deferred errors, and Uncorrected errors when ERR<n>CTLR.FI is `0b1` and *disabled* for corrected error events, Deferred errors, and Uncorrected errors when ERR<n>CTLR.FI is `0b0`.

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,FI} controls are not implemented.

A *Corrected error event* is defined as follows:

R_{CGZMD}
- If the node implements a corrected error counter 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 that 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`.

R_{XFMTV}
- If the node does not implement Corrected error counters then a corrected error event occurs when the node records an error as Corrected error.

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.

3.5 In-band error response signaling (external aborts)

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* for Uncorrected errors when ERR<n>CTLR.UE is `0b1`, and in-band error response signaling is *disabled* for Uncorrected errors 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 for Uncorrected errors.

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 records an Uncorrected error and signals an in-band error response, it sets ERR<n>STATUS.ER to `0b1`.

3.6 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. For example, if a memory controller enters a failure mode and stops servicing memory requests from application processors, and application processors host the primary error recovery software, then the error has to be signaled to a secondary error controller that has its own private resources in order to record 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 critical error as an Uncontainable error.

I_{BNDZW} Classifying the critical error condition as an Uncontainable error if 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.

3.7 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.

R_{SLPQW} For a node <n>, if a standard format corrected error counter is implemented, then the counter or counters are recorded in ERR<n>MISC0.

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

- 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.

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 the ERR<n>ADDR register then:
  - This part is valid when ERR<n>STATUS.{V, AV} == {1, 1}.
  - It is IMPLEMENTATION DEFINED whether this part of the counted-fault location is treated as valid or invalid when ERR<n>STATUS.{V, AV} == {1, 0}.
  - Otherwise, this part is invalid.

R_{JMVKQ}
- If the counted-fault location or part of the counted-fault location is held in the ERR<n>STATUS.IERR field, this part is valid when ERR<n>STATUS.V == `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, this part is valid when ERR<n>STATUS.V == `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} == {1, 1} 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} == {1, 0}.
  - Otherwise, this part is invalid.

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.

Note

- 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`.
- 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 be valid only for some values of ERR<n>STATUS.{IERR,SERR}.
- For some implementations, ERR<n>ADDR is always written when an error is recorded, meaning ERR<n>STATUS.{V, AV} == {1, 0} is never set by the hardware. Similarly, for some implementations, ERR<n>STATUS.{V, MV} == {1, 0} is never set by the hardware. For these cases the implementation might ignore the applicable one or ones of the AV and MV bits when determining whether the fault 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.

Note

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

See also:

- *Writing the error record*
- *Fault handling interrupt*

3.8 Error recovery, fault handling, and critical error signaling

I_{BHBCB} Error recovery, fault handling, and critical error interrupts are normally routed using an interrupt controller.

R_{QTQBJ} For an Arm *Generic Interrupt Controller* (GIC), if the error records of the node that generates the interrupts 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 signals for error recovery, fault handling, and critical error interrupts, or whether it shares any of these interrupts with other error records and/or other nodes.

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

R_{SVWPZ} If the fault handling interrupt is level-sensitive, it is asserted by the node for an error record <n> while any of the following apply:

- Fault handling interrupts on all Deferred errors and Uncorrected errors are enabled, the ERR<n>STATUS.V bit is `0b1`, and either or both of the ERR<n>STATUS.{DE,UE} bits are `0b1`.
- Fault handling interrupts on Corrected errors are enabled and either:
  - The node implements a corrected error counter, ERR<n>STATUS.V is `0b1`, and the counter overflow flag is `0b1`.
  - The node does not implement a corrected error counter, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.CE is nonzero.

R_{VHSRJ} If the error recovery interrupt is level-sensitive, it is asserted by the node for an error record <n> while 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_{KTVHF} If the critical error interrupt is level-sensitive, it is asserted by the node for an error record <n> while critical error interrupts are enabled, ERR<n>STATUS.V is `0b1`, and ERR<n>STATUS.CI is `0b1`.

R_{YPPWB} If the fault handling interrupt is edge-triggered, it is generated by the node for an error record when any of the following occur:

- Fault handling interrupts on all Deferred errors and Uncorrected errors are enabled, and an error is recorded in the error record as either Deferred error or Uncorrected error.
- 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, it 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 as Uncorrected error.
- Error recovery interrupts on Deferred errors are enabled, and an error is recorded in the error record as Deferred error.

R_{FLPKB} If the critical error interrupt is edge-triggered, it 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 is generated even if the ERR<n>STATUS.CI was already `0b1`.

I_{MYKYF} An enabled edge-triggered interrupt 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 is generated by a write to a register that enables an interrupt or otherwise creates the conditions for the interrupt in the other syndrome registers, as defined for a level-sensitive interrupt.

R_{GZQWV} The standard error record reserves a set of register locations for programming *Message Signaled Interrupts* (MSIs). In addition, a recommended layout for these registers is provided.

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

See also:

- ERRIRQCR<n>, ERRERICR0, ERRERICR1, ERRERICR2, ERRCRICR0, ERRCRICR1, ERRCRICR2, ERRFHICR0, ERRFHICR1, ERRFHICR2, and ERRIRQSR.
- *Synchronization and error record accesses*
- *Error recovery interrupt*
- *Fault handling interrupt*
- *Critical error interrupt*

3.9 Error recovery reset

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

The RAS System Architecture defines two classes of reset:

R_{DKKYC}
- *Cold reset* is asserted to a component when it transitions from a powered off state to a powered on state. Cold reset initializes the component to a known initial state. No state is preserved from the previous powered off state.

R_{WXRXD}
- *Error Recovery reset* is an optional reset that might be applied at any other time. System Error Recovery reset initializes the component to a known state. Unlike Cold reset, any recorded error syndrome information is preserved over a System Error Recovery reset.

R_{LPJWX} The way in which these resets map to other resets is IMPLEMENTATION DEFINED.

R_{ZLZDR} Any mechanisms for asserting resets are IMPLEMENTATION DEFINED.

I_{WFGQQ} For a PE, the Error Recovery reset might be implemented by the architectural Warm reset. If Warm reset is implemented, it preserves the error records in the PE.

3.10 Timestamp extension

R_{BWYMJ} The RAS Timestamp Extension is an optional part of RAS System Architecture v1.1.

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

R_{TRHJP} For a given error record <n>, the timestamp value is recorded in ERR<n>MISC3.

I_{FZJMJ} For a given node <n>, the RAS Timestamp Extension is implemented if ERR<n>FR.TS != `0b00`.

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

I_{HLDRQ} For a given node <n>, the value of ERR<n>FR.TS defines which timebase is used.

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.

3.11 Common Fault Injection Model Extension

R_{CVLDN} The *Common Fault Injection Model Extension* is an optional part of RAS System Architecture v1.1.

I_{TSWKX} Other fault injection mechanisms are permitted. For example, if the Common Fault Injection Model Extension is not implemented, the ERRIMPDEF<n> registers might be used for some other 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`.

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

R_{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.

I_{KHPVH} The data is not corrupted by the Common Fault Injection Model Extension.

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

I_{YSQHB} Arm recommends that the Common Fault Injection Model Extension supports all the component error state types supported by the node.

I_{QVLPN} For a given node <n>, the Common Fault Injection Model Extension is implemented if ERR<n>FR.INJ != `0b00`.

I_{ZBZHW} For a given node <n>, the Common Fault Injection Model Extension capabilities are discoverable using ERR<n>PFGF.

I_{ZDPWF} If a node is not capable of recording an component error state type, then it does not support injecting that component error state type.

R_{BQCGC} For a given node <n>, the Common Fault Injection Model Extension is disabled if ERR<n>CTLR.ED is writable and `0b0`.

I_{YMWNF} 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.

I_{QFYWD} Additional constraints might apply if fault injection can affect the operation of Secure state. See *Security and Virtualization*.

3.11.1 Operation of the Common Fault Injection Model Extension

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

R_{VDZSG} When software writes 1 to ERR<n>PFGCTL.CDNEN:

- 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 enters the fault injection state.

I_{XDWGY} The current value of the Error Generation Counter is not visible to software.

R_{PLZMT} While ERR<n>PFGCTL.CDNEN == 1 and the Error Generation Counter is nonzero, the Error Generation Counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate.

I_{DMNZX} 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.

R_{DDPMH} When the Error Generation Counter decrements to/past zero, the component enters a *fault injection state*.

R_{YXXWT} When the component is in the fault injection state, on the next access to the component, the component:

- 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, the node records the injected error.
- If error reporting and logging at the node is disabled, it is UNPREDICTABLE whether or not the node records the injected error.
- Leaves 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, the behavior is UNPREDICTABLE and is one of:

- 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 if 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 Uncorrected error or Deferred error and cannot be injected with a Corrected error.
- The ER value can be injected with an Uncorrected error and cannot be injected with a Corrected error.
- It is IMPLEMENTATION DEFINED whether the ER value can be injected with a Deferred error.

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 non-zero, 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 == 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.

Chapter 4
RAS Extension and RAS System Architecture Registers

4.1 Memory-mapped view

I_{MQDMJ} *Error record registers, including memory mapped view* defines the registers for memory-mapped 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 a group, relating to one or more nodes.

The *Reliability, Availability, Serviceability* (RAS) System Architecture defines the following reusable formats for memory-mapped error records:

R_{TPFWF}
- *Memory-mapped error record group view* describes a group of error records accessed via a standard 4KB memory-mapped peripheral.

R_{DHYDC}
- *Memory-mapped single error record view* 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, ERR<n>FR, is at an address aligned to a multiple of 64 bytes.

R_{NBFYF} In *Memory-mapped error record group view*, ERRDEVID indicates the highest numbered index of the error records that can be accessed.

I_{GFLXS} For a 4KB peripheral implementing *Memory-mapped error record group view*, up to 24 error records can be accessed if the Common Fault Injection Model Extension is implemented, and up to 56 otherwise. Groups containing more records can be defined by increasing the page size for a group. This is not described by current versions of the RAS System Architecture. For more information, contact Arm.

R_{YGWDK} In *Memory-mapped error record group view*, each error record occupies a set of locations at offsets from an error record base. This error record base is a fixed multiple of the index of the error record from the group base.

R_{YFCNK} *Memory-mapped error record group view* includes a group status register, ERRGSR.

I_{TJYGF} The Common Fault Injection Model Extension is not supported in the *Memory-mapped single error record view* format.

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 group of error records.
- Other memory-mapped components in other address spaces. For example, in both Non-secure and Secure physical address spaces.

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

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

The requirements for a memory-mapped view of RAS error records are:

R_{QRLXV}
- Reads and writes of unallocated locations are reserved accesses.

R_{PPFBS}
- Reads and writes of locations for features that are not implemented are reserved accesses, including:
  - OPTIONAL features that are not implemented.
  - error records that are not implemented.

R_{BNKVL}
- Reads of WO locations are reserved accesses.

R_{WNFYH}

- Writes to RO locations are reserved accesses.

R_{RZWDM}

Reserved accesses are RAZ/WI. However, software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture. Software must treat reserved accesses as RES0.

R_{JXHNT}

The memory access sizes that are supported by the memory-mapped component are as described for other memory-mapped components in [1]. It is IMPLEMENTATION DEFINED whether a word-aligned 32-bit access to either half of a doubleword-aligned 64-bit register is supported if there is no *Processing element* (PE) in the system that supports AArch32.

4.2 Reset values

IPQVFQ 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, this is because one of the following applies, and it is IMPLEMENTATION DEFINED which:

- 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 Accessibility in ERR<n>STATUS.

4.3 Error record registers, including memory mapped view

INFQQQ This section describes the error record registers. The descriptions in this section apply whether the error record is accessed:

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

4.3.1 Register index

4.3.1.1 Using AArch32 System registers

Table 4.1: Using AArch32 System registers, System register map

Use	To Access	Access	Description
ERXADDR	ERR<n>ADDR[31:0]	R/W	Error Record Address Register
ERXADDR2	ERR<n>ADDR[63:32]	R/W	Error Record Address Register
ERXCTLR	ERR<n>CTLR[31:0]	R/W	Error Record Control Register
ERXCTLR2	ERR<n>CTLR[63:32]	R/W	Error Record Control Register
ERXFR	ERR<n>FR[31:0]	RO	Error Record Feature Register
ERXFR2	ERR<n>FR[63:32]	RO	Error Record Feature Register
ERXMISC0	ERR<n>MISC0[31:0]	R/W	Error Record Miscellaneous Register 0
ERXMISC1	ERR<n>MISC0[63:32]	R/W	Error Record Miscellaneous Register 0
ERXMISC2	ERR<n>MISC1[31:0]	R/W	Error Record Miscellaneous Register 1
ERXMISC3	ERR<n>MISC1[63:32]	R/W	Error Record Miscellaneous Register 1
ERXMISC4	ERR<n>MISC2[31:0]	R/W	Error Record Miscellaneous Register 2
ERXMISC5	ERR<n>MISC2[63:32]	R/W	Error Record Miscellaneous Register 2
ERXMISC6	ERR<n>MISC3[31:0]	R/W	Error Record Miscellaneous Register 3
ERXMISC7	ERR<n>MISC3[63:32]	R/W	Error Record Miscellaneous Register 3
ERXSTATUS	ERR<n>STATUS[31:0]	R/W	Error Record Primary Status Register

4.3.1.2 Using AArch64 System registers

Table 4.2: Using AArch64 System registers, System register map

Use	To Access	Access	Description
ERXADDR_EL1	ERR<n>ADDR	R/W	Error Record Address Register
ERXCTLR_EL1	ERR<n>CTLR	R/W	Error Record Control Register
ERXFR_EL1	ERR<n>FR	RO	Error Record Feature Register
ERXMISC0_EL1	ERR<n>MISC0	R/W	Error Record Miscellaneous Register 0
ERXMISC1_EL1	ERR<n>MISC1	R/W	Error Record Miscellaneous Register 1
ERXMISC2_EL1	ERR<n>MISC2	R/W	Error Record Miscellaneous Register 2
ERXMISC3_EL1	ERR<n>MISC3	R/W	Error Record Miscellaneous Register 3
ERXPFGCDN_EL1	ERR<n>PFGCDN	R/W	Pseudo-fault Generation Countdown Register
ERXPFGCTL_EL1	ERR<n>PFGCTL	R/W	Pseudo-fault Generation Control Register
ERXPFGF_EL1	ERR<n>PFGF	RO	Pseudo-fault Generation Feature Register
ERXSTATUS_EL1	ERR<n>STATUS	R/W	Error Record Primary Status Register

4.3.1.3 Memory-mapped error record group view

Table 4.3: RAS, error record group, memory-mapped register map

Offset	Access	Size	Register	Description
0x000+64×*n*	RO	64	ERR<n>FR	Error Record Feature Register
0x008+64×*n*	R/W	64	ERR<n>CTLR	Error Record Control Register
0x010+64×*n*	R/W	64	ERR<n>STATUS	Error Record Primary Status Register
0x018+64×*n*	R/W	64	ERR<n>ADDR	Error Record Address Register
0x020+64×*n*	R/W	64	ERR<n>MISC0	Error Record Miscellaneous Register 0
0x028+64×*n*	R/W	64	ERR<n>MISC1	Error Record Miscellaneous Register 1
0x030+64×*n*	R/W	64	ERR<n>MISC2	Error Record Miscellaneous Register 2
0x038+64×*n*	R/W	64	ERR<n>MISC3	Error Record Miscellaneous Register 3
0x800+64×*n*	RO	64	ERR<n>PFGF	Pseudo-fault Generation Feature Register
0x800+8×*n*	R/W	64	ERRIMPDEF<n>	IMPLEMENTATION DEFINED Register <*n*>
0x808+64×*n*	R/W	64	ERR<n>PFGCTL	Pseudo-fault Generation Control Register
0x810+64×*n*	R/W	64	ERR<n>PFGCDN	Pseudo-fault Generation Countdown Register
0xE00	RO	64	ERRGSR	Error Group Status Register
0xE10	RO	32	ERRIIDR	Implementation Identification Register
0xE80	R/W	64	ERRFHICR0	Fault Handling Interrupt Configuration Register 0
0xE80+8×*n*	R/W	64	ERRIRQCR<n>	Generic Error Interrupt Configuration Register
0xE88	R/W	32	ERRFHICR1	Fault Handling Interrupt Configuration Register 1
0xE8C	R/W	32	ERRFHICR2	Fault Handling Interrupt Configuration Register 2
0xE90	R/W	64	ERRERICR0	Error Recovery Interrupt Configuration Register 0
0xE98	R/W	32	ERRERICR1	Error Recovery Interrupt Configuration Register 1
0xE9C	R/W	32	ERRERICR2	Error Recovery Interrupt Configuration Register 2
0xEA0	R/W	64	ERRCRICR0	Critical Error Interrupt Configuration Register 0
0xEA8	R/W	32	ERRCRICR1	Critical Error Interrupt Configuration Register 1
0xEAC	R/W	32	ERRCRICR2	Critical Error Interrupt Configuration Register 2
0xEF8	R/W	64	ERRIRQSR	Error Interrupt Status Register
0xFA8	RO	64	ERRDEVAFF	Device Affinity Register
0xFBC	RO	32	ERRDEVARCH	Device Architecture Register
0xFC8	RO	32	ERRDEVID	Device Configuration Register
0xFD0	RO	32	ERRPIDR4	Peripheral Identification Register 4
0xFE0	RO	32	ERRPIDR0	Peripheral Identification Register 0
0xFE4	RO	32	ERRPIDR1	Peripheral Identification Register 1
0xFE8	RO	32	ERRPIDR2	Peripheral Identification Register 2
0xFEC	RO	32	ERRPIDR3	Peripheral Identification Register 3
0xFF0	RO	32	ERRCIDR0	Component Identification Register 0
0xFF4	RO	32	ERRCIDR1	Component Identification Register 1
0xFF8	RO	32	ERRCIDR2	Component Identification Register 2
0xFFC	RO	32	ERRCIDR3	Component Identification Register 3

4.3.1.4 Memory-mapped single error record view

Table 4.4: RAS, single error record, memory-mapped register map

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

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

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.

Configurations

ERR<*n*>ADDR is present only if all of the following are true:

- Error record <*n*> is implemented.
- Error record <*n*> includes an address associated with an error.

ERR<*n*>ADDR is RES0 otherwise.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

Attributes

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

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

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

4.3.2.1 Field descriptions

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

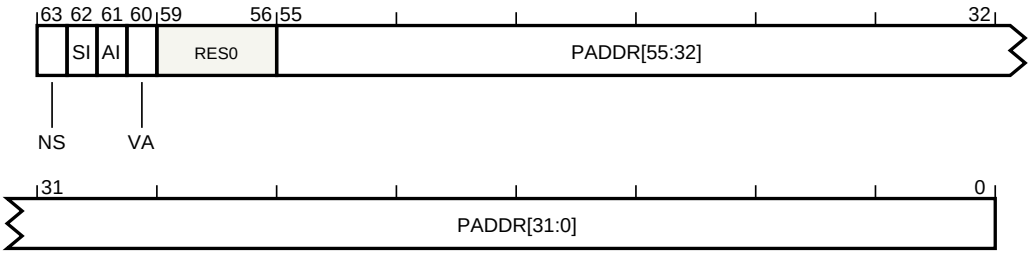

Figure 4.1: ERR<*n*>ADDR

NS, bit [63]

Non-secure attribute.

The possible values of this bit are:

`0`	ERR<*n*>ADDR.PADDR is a Secure address.
`1`	ERR<*n*>ADDR.PADDR is a Non-secure address.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

SI, bit [62]
Secure Incorrect.

Indicates whether ERR<*n*>ADDR.NS is valid. The possible values of this bit are:

0	ERR<*n*>ADDR.NS is correct. That is, it matches the programmers' view of the Non-secure attribute for the recorded location.
1	ERR<*n*>ADDR.NS might not be correct, and might not match the programmers' view of the Non-secure attribute for the recorded location.

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

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

AI, bit [61]
Address Incorrect. Indicates whether ERR<*n*>ADDR.PADDR is a valid physical address that is known to match the programmers' view of the physical address for the recorded location. The possible values of this bit are:

0	ERR<*n*>ADDR.PADDR is a valid physical address. That is, it matches the programmers' view of the physical address for the recorded location.
1	ERR<*n*>ADDR.PADDR might not be a valid physical address, and might not match the programmers' view of the physical address for the recorded location.

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

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

VA, bit [60]
Virtual Address. Indicates whether ERR<*n*>ADDR.PADDR field is a virtual address. The possible values of this bit are:

0	ERR<*n*>ADDR.PADDR is not a virtual address.
1	ERR<*n*>ADDR.PADDR is a virtual address.

No context information is provided for the virtual address. When ERR<*n*>ADDR.VA == 0b1, ERR<*n*>ADDR.{NS,SI,AI} read as {0,1,1}.

Support for this bit is optional. If this bit 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}.

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

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Bits [59:56]
Reserved. This field is RES0.

PADDR, bits [55:0]
Physical Address. Address of the recorded location. If the physical address size implemented by this component is smaller than the size of this field, then high-order bits are unimplemented and either RES0 or have a fixed read-only IMPLEMENTATION DEFINED value. Low-order address bits might also be

unimplemented and RES0, for example, if the physical address is always aligned to the size of a protection granule.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

4.3.2.2 Accessibility

ERR<*n*>ADDR ignores writes if all of the following are true:

- Any of the following are true:
  - The RAS Common Fault Injection Model Extension is implemented by the node that owns this error record and ERR<q>PFGF.AV == `0b0`.
  - The RAS Common Fault Injection Model Extension is not implemented by the node that owns this error record.
- ERR<n>STATUS.AV == `0b1`.

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

The ERR<*n*>CTLR characteristics are:

Purpose

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

- Enabling error detection and correction.
- Enabling 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.

Configurations

ERR<*n*>CTLR is present only if all of the following are true:

- Error record <*n*> is implemented.
- Error record <*n*> is the first error record owned by a node.

ERR<*n*>CTLR is RES0 otherwise.

ERR<n>FR describes the features implemented by the node.

Attributes

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

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

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

4.3.3.1 Field descriptions

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

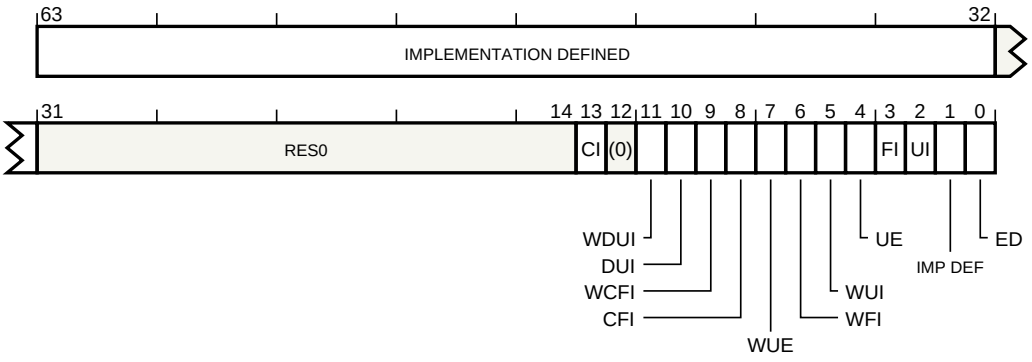

Figure 4.2: ERR<*n*>CTLR

Bits [63:32]

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

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

Bits [31:14,12]

Reserved. This field is RES0.

CI, bit [13]

Critical error interrupt enable.

When ERR<*n*>FR.CI == 0b10

When enabled, the critical error interrupt is generated for a critical error condition. The possible values of this bit are:

0	Critical error interrupt not generated for critical errors. Critical errors are treated as Uncontained errors.
1	Critical error interrupt generated for critical errors.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

WDUI, bit [11]

Error recovery interrupt for deferred errors on writes enable.

When ERR<*n*>FR.DUI == 0b11

When enabled, the error recovery interrupt is generated for detected Deferred errors on writes.

The possible values of this bit are:

0	Error recovery interrupt not generated for deferred errors on writes.
1	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.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

DUI, bit [10]

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

When ERR<*n*>FR.DUI == 0b10

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 detected Deferred errors.

The possible values of this bit are:

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

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

When ERR<*n*>FR.DUI == 0b11

Error recovery interrupt for deferred errors on reads enable.

When ERR<n>FR.DUI == 0b11, this bit is named RDUI.

When enabled, the error recovery interrupt is generated for detected Deferred errors on reads.

The possible values of this bit are:

0	Error recovery interrupt not generated for deferred errors on reads.
1	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.

WCFI, bit [9]

Fault handling interrupt for Corrected errors on writes enable.

When ERR<*n*>FR.CFI == 0b11

When enabled:

- If the node implements Corrected error counters for writes, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 0b1. For more information, see ERR<n>MISC0.
- Otherwise, the fault handling interrupt is also generated for detected Corrected errors on writes.

The possible values of this bit are:

0	Fault handling interrupt not generated for Corrected errors on writes.
1	Fault handling interrupt generated for Corrected errors on writes.

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

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

CFI, bit [8]

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

When ERR<*n*>FR.CFI == 0b10

Fault handling interrupt for Corrected errors enable.

When ERR<n>FR.CFI == 0b10, this control applies to errors arising from both reads and writes.

When enabled:

- If the node implements Corrected error counters, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 0b1. For more information, see ERR<n>MISC0.
- Otherwise, the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

0	Fault handling interrupt not generated for Corrected errors.
1	Fault handling interrupt generated for Corrected errors.

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

When ERR<*n*>FR.CFI == 0b11
Fault handling interrupt for Corrected errors on reads enable.

When ERR<n>FR.CFI == `0b11`, this bit is named RCFI.

When enabled:

- If the node implements Corrected error counters for reads, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to `0b1`. For more information, see ERR<n>MISC0.
- Otherwise, the fault handling interrupt is also generated for detected Corrected errors on reads.

The possible values of this bit are:

`0`	Fault handling interrupt not generated for Corrected errors on reads.
`1`	Fault handling interrupt generated for Corrected errors on reads.

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

WUE, bit [7]
In-band Uncorrected error reporting on writes enable.

When ERR<*n*>FR.UE == 0b11
When enabled, responses to writes that detect an Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External Abort).

The possible values of this bit are:

`0`	External Abort response for Uncorrected errors on writes disabled.
`1`	External Abort response for Uncorrected errors on writes enabled.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

WFI, bit [6]
Fault handling interrupt on writes enable.

When ERR<*n*>FR.FI == 0b11
When enabled:

- The fault handling interrupt is generated for detected Deferred errors and Uncorrected errors.
- If the corresponding fault handling interrupt for Corrected errors control is not implemented:
  - If the node implements Corrected error counters for writes, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to `0b1`.
  - Otherwise, the fault handling interrupt is also generated for detected Corrected errors on writes.

The possible values of this bit are:

`0`	Fault handling interrupt on writes disabled.
`1`	Fault handling 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.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

WUI, bit [5]
Uncorrected error recovery interrupt on writes enable.

When ERR<*n*>FR.UI == 0b11
When enabled, the error recovery interrupt is generated for detected Uncorrected errors on writes that are not deferred.

The possible values of this bit are:

0	Error recovery interrupt on writes disabled.
1	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.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

UE, bit [4]
This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

When ERR<*n*>FR.UE == 0b10
In-band Uncorrected error reporting 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 Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External Abort).

The possible values of this bit are:

0	External Abort response for Uncorrected errors disabled.
1	External Abort response for Uncorrected errors enabled.

When ERR<*n*>FR.UE == 0b11
In-band Uncorrected error reporting on reads enable.

When ERR<n>FR.UE == 0b11, this bit is named RUE.

When enabled, responses to reads that detect an Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External Abort).

The possible values of this bit are:

0	External Abort response for Uncorrected errors on reads disabled.
1	External Abort response for Uncorrected errors on reads enabled.

FI, bit [3]

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

When ERR<*n*>FR.FI == 0b10

Fault handling interrupt enable.

When ERR<n>FR.FI == `0b10`, this control applies to errors arising from both reads and writes.

When enabled:

- The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors.
- If the fault handling interrupt for Corrected errors control is not implemented:
  - If the node implements Corrected error counters, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to `0b1`.
  - Otherwise, the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

`0`	Fault handling interrupt disabled.
`1`	Fault handling interrupt enabled.

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

When ERR<*n*>FR.FI == 0b11

Fault handling interrupt on reads enable.

When ERR<n>FR.FI == `0b11`, this bit is named RFI.

When enabled:

- The fault handling interrupt is generated for detected Deferred errors and Uncorrected errors.
- If the corresponding fault handling interrupt for Corrected errors control is not implemented:
  - If the node implements Corrected error counters for reads, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to `0b1`.
  - Otherwise, the fault handling interrupt is also generated for detected Corrected errors on reads.

The possible values of this bit are:

`0`	Fault handling interrupt on reads disabled.
`1`	Fault handling 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.

UI, bit [2]

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

When ERR<*n*>FR.UI == 0b10

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 detected Uncorrected errors that are not deferred.

The possible values of this bit are:

0	Error recovery interrupt disabled.
1	Error recovery interrupt enabled.

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

When ERR<*n*>FR.UI == 0b11

Uncorrected error recovery interrupt on reads enable.

When ERR<n>FR.UI == 0b11, this bit is named RUI.

When enabled, the error recovery interrupt is generated for detected Uncorrected errors on reads that are not deferred.

The possible values of this bit are:

0	Error recovery interrupt on reads disabled.
1	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.

Bit [1]

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

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

ED, bit [0]

Error reporting and logging enable.

When ERR<*n*>FR.ED == 0b10

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. The possible values of this bit are:

0	Error reporting disabled.
1	Error reporting enabled.

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting is disabled. That is, even with error reporting disabled, the node might continue to silently correct errors. Uncorrectable errors might result in corrupt data being silently propagated by the node.

This bit resets to an IMPLEMENTATION DEFINED value on a Cold reset. This bit is preserved on an Error Recovery reset.

Note:

If this node requires initialization after Cold reset to prevent signaling false errors, then Arm recommends this bit is set to 0b0 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 0b1, the reset values of other controls in this register are also IMPLEMENTATION DEFINED and should not be UNKNOWN.

Otherwise
Reserved. This bit is RES0.

4.3.3.2 Accessibility

None.

4.3.4 ERR<*n*>FR, Error Record Feature Register

The ERR<*n*>FR characteristics are:

Purpose

Defines whether <*n*> is the first record owned by a node:

- If <*n*> is the first error record owned by a node, then ERR<*n*>FR.ED != `0b00`.
- If <*n*> is not the first error record owned by a node, then ERR<*n*>FR.ED == `0b00`.

If <*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.

Configurations

ERR<*n*>FR is present only if error record <*n*> is implemented. ERR<*n*>FR is RES0 otherwise.

Attributes

When accessed using a System register, ERR<*n*>FR is a 64-bit read-only register accessed using:

- `MRC` of ERXFR for ERR<*n*>FR[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` of ERXFR2 for ERR<*n*>FR[63:32] when ERRSELR.SEL is set to *n*.
- `MRS` of ERXFR_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>FR is a 64-bit read-only register located at offset `0x000` + 64×*n*.

4.3.4.1 ERR<*n*>FR (ERR<*n*>FR.ED != `0b00`)

The ERR<*n*>FR (ERR<*n*>FR.ED != `0b00`) bit assignments are:

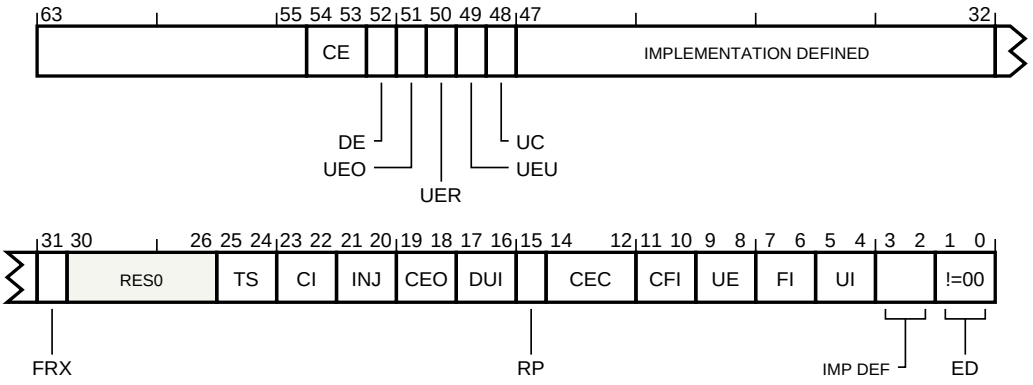

Figure 4.3: ERR<*n*>FR

Bits [63:55]

Reserved.

When ERR<*n*>FR.FRX == 0b1

Reserved. This field is RES0.

Otherwise

Reserved for identifying IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value.

CE, bits [54:53]

Corrected Error recording.

When ERR<*n*>FR.FRX == 0b1

Describes the types of Corrected Error the node can record. The defined values of this field are:

0b00	The node does not record any type of Corrected Error.
0b01	The node can record transient or persistent Corrected Errors (Corrected Errors that are recorded as ERR<n>STATUS.CE == 0b01 and 0b11).
0b10	The node can record of a non-specific Corrected Error (a Corrected Error that is recorded as ERR<n>STATUS.CE == 0b10).
0b11	The node can record any type of Corrected Error.

Otherwise
Reserved for identifying IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value.

DE, bit [52]
Deferred Error recording.

When ERR<*n*>FR.FRX == 0b1
Describes whether the node can record this type of error. The defined values of this bit are:

0	The node does not record this type of error.
1	The node can record this type of error.

Otherwise
Reserved for identifying IMPLEMENTATION DEFINED controls. This bit reads as an IMPLEMENTATION DEFINED value.

UEO, bit [51]
Latent or Restartable Error recording.

When ERR<*n*>FR.FRX == 0b1
Describes whether the node can record this type of error. The defined values of this bit are:

0	The node does not record this type of error.
1	The node can record this type of error.

Otherwise
Reserved for identifying IMPLEMENTATION DEFINED controls. This bit reads as an IMPLEMENTATION DEFINED value.

UER, bit [50]
Signaled or Recoverable Error recording.

When ERR<*n*>FR.FRX == 0b1
Describes whether the node can record this type of error. The defined values of this bit are:

0	The node does not record this type of error.
1	The node can record this type of error.

Otherwise
Reserved for identifying IMPLEMENTATION DEFINED controls. This bit reads as an IMPLEMENTATION DEFINED value.

UEU, bit [49]
Unrecoverable Error recording.

When ERR<*n*>FR.FRX == 0b1

Describes whether the node can record this type of error. The defined values of this bit are:

0	The node does not record this type of error.
1	The node can record this type of error.

Otherwise

Reserved for identifying IMPLEMENTATION DEFINED controls. This bit reads as an IMPLEMENTATION DEFINED value.

UC, bit [48]

Uncontainable Error recording.

When ERR<*n*>FR.FRX == 0b1

Describes whether the node can record this type of error. The defined values of this bit are:

0	The node does not record this type of error.
1	The node can record this type of error.

Otherwise

Reserved for identifying IMPLEMENTATION DEFINED controls. This bit reads as an IMPLEMENTATION DEFINED value.

Bits [47:32]

Reserved for identifying IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value.

FRX, bit [31]

Feature Register extension.

When RAS System Architecture v1.1 is implemented

Defines whether ERR<*n*>FR[63:48] are architecturally defined. The defined values of this bit are:

0b0	ERR<*n*>FR[63:48] are IMPLEMENTATION DEFINED.
0b1	ERR<*n*>FR[63:48] are defined by the architecture.

Otherwise

Reserved. This bit is RES0.

Bits [30:26]

Reserved. This field is 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 defined values of this field are:

0b00	The node does not support a timestamp register.
0b01	The node implements a timestamp register. The timestamp uses the same timebase as the system Generic Timer. **Note:** For an error record which 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	The node implements a timestamp register. The timebase for the timestamp is IMPLEMENTATION DEFINED.

All other values are reserved.

CI, bits [23:22]
Critical error interrupt. Indicates whether the critical error interrupt and associated controls are implemented. The defined values of this field are:

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.

INJ, bits [21:20]
Fault Injection Extension. Indicates whether the RAS Common Fault Injection Model Extension is implemented. The defined values of this field are:

0b00	The node does not support the RAS Common Fault Injection Model Extension.
0b01	The node implements the RAS Common Fault Injection Model Extension. See ERR<n>PFGF for more information.

All other values are reserved.

CEO, bits [19:18]
Corrected Error overwrite.

When ERR<*n*>FR.CEC != 0b000
Indicates the behavior when a second Corrected error is detected after a first Corrected error has been recorded by an error record <*m*> owned by the node. The defined values of this field are:

0b00	Counts Corrected errors if a counter is implemented. Keeps the previous error syndrome. If the counter overflows, or no counter is implemented, then ERR<m>STATUS.OF is set to 0b1.
0b01	Counts Corrected errors. If ERR<m>STATUS.OF == 0b1 before the Corrected error is counted, then keeps the previous syndrome. Otherwise the previous syndrome is overwritten. If the counter overflows, then ERR<m>STATUS.OF is set to 0b1.

All other values are reserved.

See Writing the error record.

Otherwise
Reserved. This field is RES0.

DUI, bits [17:16]
Error recovery interrupt for deferred errors control.

When ERR<*n*>FR.UI != 0b00
Indicates whether the control for enabling error recovery interrupts on deferred errors are implemented. The defined values of this field are:

0b00	Does not support the control for enabling error recovery interrupts on deferred errors. ERR<n>CTLR.DUI is RES0.
0b10	Control for enabling error recovery interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DUI.
0b11	Control for enabling 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.

Otherwise
Reserved. This field is RES0.

RP, bit [15]
Repeat counter.

When ERR<*n*>FR.CEC != 0b000
Indicates whether the node implements the repeat Corrected error counter in ERR<m>MISC0 for each error record <*m*> owned by the node that implements the standard Corrected error counter. The defined values of this bit are:

0	A single CE counter is implemented.
1	A first (repeat) counter and a second (other) counter are implemented. The repeat counter is the same size as the primary error counter.

Otherwise
Reserved. This bit is RES0.

CEC, bits [14:12]
Corrected Error Counter. Indicates whether the node implements the standard Corrected error counter (CE counter) mechanisms in ERR<m>MISC0 for each error record <*m*> owned by the node that can record countable errors. The defined values of this field are:

0b000	Does not implement the standard Corrected error counter model.
0b010	Implements an 8-bit Corrected error counter in ERR<m>MISC0[39:32].
0b100	Implements a 16-bit Corrected error counter in ERR<m>MISC0[47:32].

All other values are reserved.

Note:

Implementations might include other error counter models, or might include the standard model and not indicate this in ERR<*n*>FR.

CFI, bits [11:10]
Fault handling interrupt for corrected errors.

When ERR<*n*>FR.FI != 0b00
Indicates whether the control for enabling fault handling interrupts on corrected errors are implemented. The defined values of this field are:

0b00	Does not support the control for enabling fault handling interrupts on corrected errors. ERR<n>CTLR.CFI is RES0.
0b10	Control for enabling fault handling interrupts on corrected errors is supported and controllable using ERR<n>CTLR.CFI.

0b11	Control for enabling 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.

Otherwise
Reserved. This field is RES0.

UE, bits [9:8]
In-band uncorrected error reporting. Indicates whether the in-band uncorrected error reporting (External Aborts) and associated controls are implemented. The defined values of this field are:

0b00	Does not support the in-band uncorrected error reporting (External Aborts). ERR<n>CTLR.UE is RES0.
0b01	In-band uncorrected error reporting (External Aborts) is supported and always enabled. ERR<n>CTLR.UE is RES0.
0b10	In-band uncorrected error reporting (External Aborts) is supported and controllable using ERR<n>CTLR.UE.
0b11	In-band uncorrected error reporting (External Aborts) is supported and controllable using ERR<n>CTLR.WUE for writes and ERR<n>CTLR.RUE for reads.

FI, bits [7:6]
Fault handling interrupt. Indicates whether the fault handling interrupt and associated controls are implemented. The defined values of this field are:

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.

UI, bits [5:4]
Error recovery interrupt for uncorrected errors. Indicates whether the error handling interrupt and associated controls are implemented. The defined values of this field are:

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.

Bits [3:2]
This field reads as an IMPLEMENTATION DEFINED value.

ED, bits [1:0]
Error reporting and logging. Indicates whether error record <*n*> is the first record owned the node, and, if so, whether it implements the controls for enabling and disabling error reporting and logging. The defined values of this field are:

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.

4.3.4.2 ERR<*n*>FR (ERR<*n*>FR.ED == 0b00)

The ERR<*n*>FR (ERR<*n*>FR.ED == 0b00) bit assignments are:

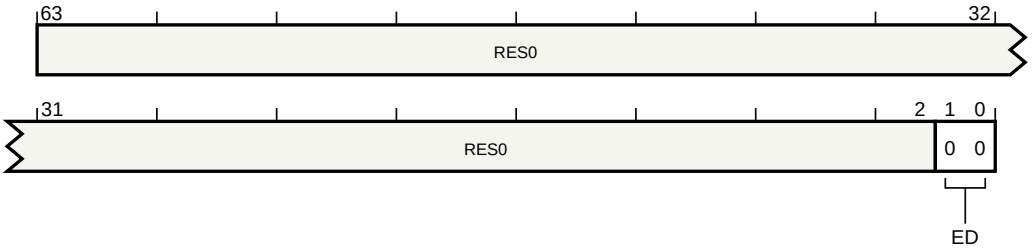

Figure 4.4: ERR<*n*>FR

Bits [63:2]
Reserved. This field is RES0.

ED, bits [1:0]
Error reporting and logging. Indicates error record <*n*> is not the first record owned the node. The defined values of this field are:

0b00	Error record <*n*> is not the first record owned by the node.

This field reads as 0b00.

4.3.4.3 Accessibility

None.

4.3.5 ERR<*n*>MISC0, Error Record Miscellaneous Register 0

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 *Field Replaceable Unit* (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 architecturally-defined error counters (ERR<q>FR.CEC != `0b000`), and error record <*n*> can record countable errors, then ERR<*n*>MISC0 implements the architecturally-defined error counter or counters.

Configurations

ERR<*n*>MISC0 is present only if error record <*n*> is implemented. ERR<*n*>MISC0 is RES0 otherwise.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

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, non-zero, 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 ERR<q>CTLR.

Attributes

When accessed using a System register, ERR<*n*>MISC0 is a 64-bit read/write register accessed using:

- `MRC` and `MCR` of ERXMISC0 for ERR<*n*>MISC0[31:0] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXMISC0_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC1 for ERR<*n*>MISC0[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC0 is a 64-bit read/write register located at offset `0x020` + 64×*n*.

4.3.5.1 ERR<*n*>MISC0 (ERR<*q*>FR.CEC == `0b000`)

The ERR<*n*>MISC0 (ERR<*q*>FR.CEC == `0b000`) bit assignments are:

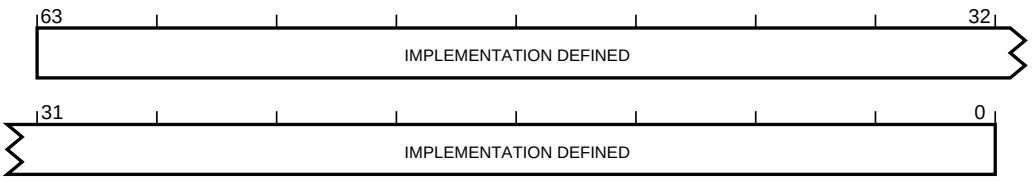

Figure 4.5: ERR<*n*>MISC0

Bits [63:0]

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

4.3.5.2 ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b100 && ERR<*q*>FR.RP == 0b0)

The ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b100 && ERR<*q*>FR.RP == 0b0) bit assignments are:

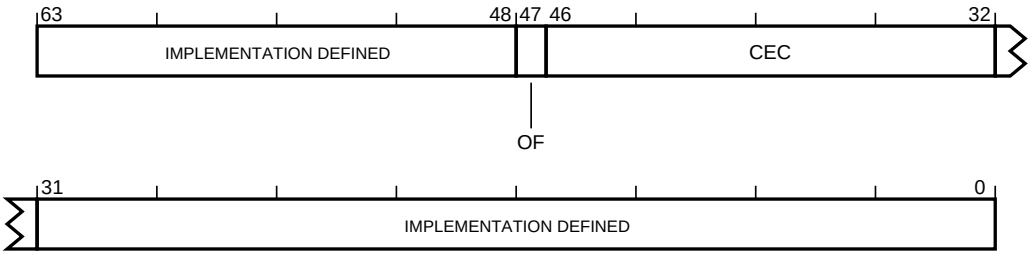

Figure 4.6: ERR<*n*>MISC0

Bits [63:48,31:0]

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

OF, bit [47]

Sticky overflow bit. Set to 1 when ERR<*n*>MISC0.CEC is incremented and wraps through zero. The possible values of this bit are:

0	Counter has not overflowed.
1	Counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

4.3.5.3 ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b010 && ERR<*q*>FR.RP == 0b0)

The ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b010 && ERR<*q*>FR.RP == 0b0) bit assignments are:

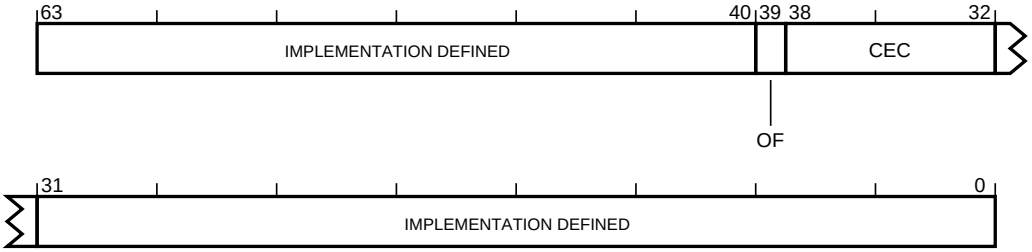

Figure 4.7: ERR<*n*>MISC0

Bits [63:40,31:0]
IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OF, bit [39]
Sticky overflow bit. Set to 1 when ERR<*n*>MISC0.CEC is incremented and wraps through zero. The possible values of this bit are:

`0`	Counter has not overflowed.
`1`	Counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

4.3.5.4 ERR<*n*>MISC0 (ERR<*q*>FR.CEC == `0b100` && ERR<*q*>FR.RP == `0b1`)

The ERR<*n*>MISC0 (ERR<*q*>FR.CEC == `0b100` && ERR<*q*>FR.RP == `0b1`) bit assignments are:

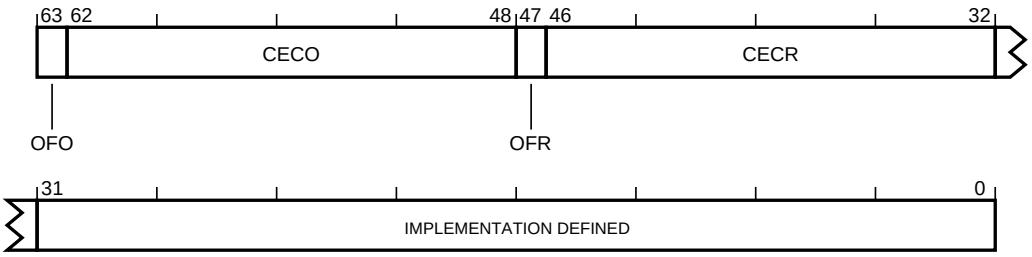

Figure 4.8: ERR<*n*>MISC0

OFO, bit [63]
Sticky overflow bit, other. Set to 1 when ERR<*n*>MISC0.CECO is incremented and wraps through zero. The possible values of this bit are:

0	Other counter has not overflowed.
1	Other counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

CECO, bits [62:48]
Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<*n*>MISC0.CECR.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

OFR, bit [47]
Sticky overflow bit, repeat. Set to 1 when ERR<*n*>MISC0.CECR is incremented and wraps through zero. The possible values of this bit are:

0	Repeat counter has not overflowed.
1	Repeat counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

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.

Bits [31:0]
IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

4.3.5.5 ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b010 && ERR<*q*>FR.RP == 0b1)

The ERR<*n*>MISC0 (ERR<*q*>FR.CEC == 0b010 && ERR<*q*>FR.RP == 0b1) bit assignments are:

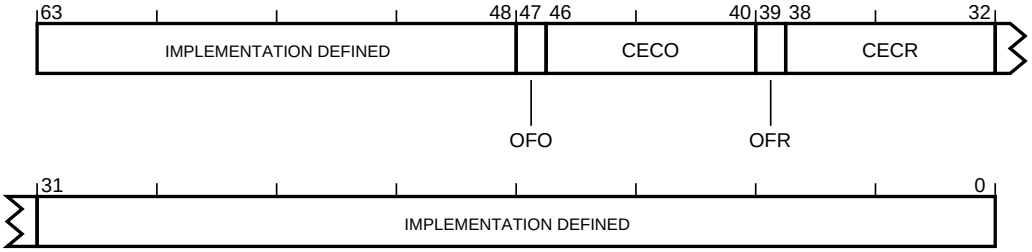

Figure 4.9: ERR<*n*>MISC0

Bits [63:48,31:0]
IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OFO, bit [47]
Sticky overflow bit, other. Set to 1 when ERR<*n*>MISC0.CECO is incremented and wraps through zero. The possible values of this bit are:

0	Other counter has not overflowed.
1	Other counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

CECO, bits [46:40]
Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<*n*>MISC0.CECR.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

OFR, bit [39]
Sticky overflow bit, repeat. Set to 1 when ERR<*n*>MISC0.CECR is incremented and wraps through zero. The possible values of this bit are:

0	Repeat counter has not overflowed.
1	Repeat counter has overflowed.

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

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.

4.3.5.6 Accessibility

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 ERR<q>PFGF.MV is `0b1`, then some parts of this register are read/write when ERR<n>STATUS.MV == `0b1`. 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 == `0b1`, 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.

4.3.6 ERR<*n*>MISC1, Error Record Miscellaneous Register 1

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.

Configurations

ERR<*n*>MISC1 is present only if error record <*n*> is implemented. ERR<*n*>MISC1 is RES0 otherwise.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

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, non-zero, 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 ERR<q>CTLR.

Attributes

When accessed using a System register, ERR<*n*>MISC1 is a 64-bit read/write register accessed using:

- `MRS` and `MSR` of ERXMISC1_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC2 for ERR<*n*>MISC1[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC3 for ERR<*n*>MISC1[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC1 is a 64-bit read/write register located at offset `0x028` + 64×*n*.

4.3.6.1 Field descriptions

The ERR<*n*>MISC1 bit assignments are:

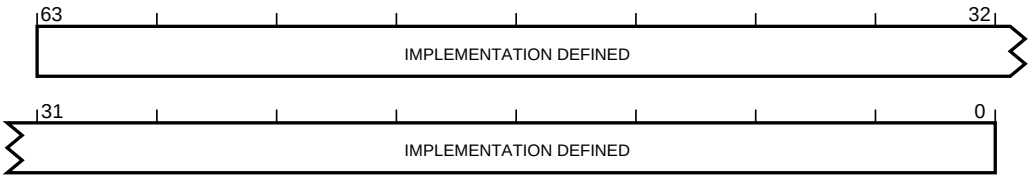

Figure 4.10: ERR<*n*>MISC1

Bits [63:0]

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

4.3.6.2 Accessibility

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 ERR<q>PFGF.MV is `0b1`, then some parts of this register are read/write when ERR<n>STATUS.MV == `0b1`. 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 == `0b1`, 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.

4.3.7 ERR<*n*>MISC2, Error Record Miscellaneous Register 2

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.

Configurations

ERR<*n*>MISC2 is present if error record <*n*> is implemented. It is IMPLEMENTATION DEFINED whether ERR<*n*>MISC2 is present if RAS System Architecture v1.1 is not implemented. ERR<*n*>MISC2 is RES0 if not present.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

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, non-zero, and ignore writes are compliant with this requirement.

If RAS System Architecture v1.1 is not implemented, Arm recommends that 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 ERR<q>CTLR.

Attributes

When accessed using a System register, ERR<*n*>MISC2 is a 64-bit read/write register accessed using:

- `MRS` and `MSR` of ERXMISC2_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC4 for ERR<*n*>MISC2[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC5 for ERR<*n*>MISC2[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC2 is a 64-bit read/write register located at offset `0x030` + 64×*n*.

4.3.7.1 Field descriptions

The ERR<*n*>MISC2 bit assignments are:

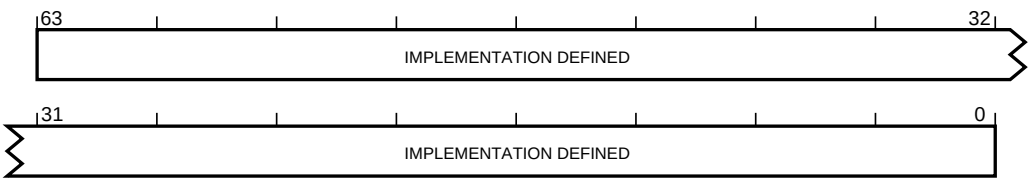

Figure 4.11: ERR<*n*>MISC2

Bits [63:0]

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

4.3.7.2 Accessibility

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 ERR<q>PFGF.MV is `0b1`, then some parts of this register are read/write when ERR<n>STATUS.MV == `0b1`. 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 == `0b1`, 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.

4.3.8 ERR<*n*>MISC3, Error Record Miscellaneous Register 3

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 (ERR<q>FR.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.

Configurations

ERR<*n*>MISC3 is present if error record <*n*> is implemented. It is IMPLEMENTATION DEFINED whether ERR<*n*>MISC3 is present if RAS System Architecture v1.1 is not implemented. ERR<*n*>MISC3 is RES0 if not present.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

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, non-zero, and ignore writes are compliant with this requirement.

If RAS System Architecture v1.1 is not implemented, Arm recommends that 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 ERR<q>CTLR.

Attributes

When accessed using a System register, ERR<*n*>MISC3 is a 64-bit read/write register accessed using:

- `MRS` and `MSR` of ERXMISC3_EL1 when ERRSELR_EL1.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC6 for ERR<*n*>MISC3[31:0] when ERRSELR.SEL is set to *n*.
- `MRC` and `MCR` of ERXMISC7 for ERR<*n*>MISC3[63:32] when ERRSELR.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>MISC3 is a 64-bit read/write register located at offset `0x038` + 64×*n*.

4.3.8.1 ERR<*n*>MISC3 (ERR<*q*>FR.TS != `0b00`)

The ERR<*n*>MISC3 (ERR<*q*>FR.TS != `0b00`) bit assignments are:

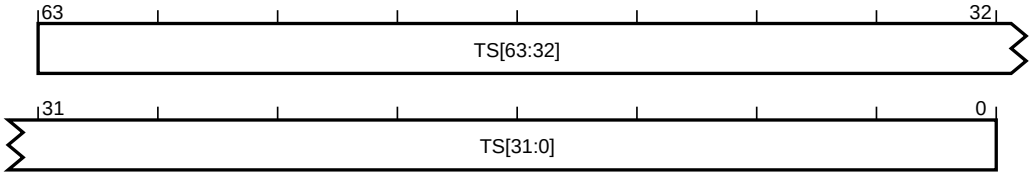

Figure 4.12: ERR<*n*>MISC3

TS, bits [63:0]

Timestamp. Timestamp value recorded when the error was detected. Valid only if ERR<n>STATUS.V == 0b1.

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

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

See ERR<q>FR.TS.

4.3.8.2 ERR<*n*>MISC3 (ERR<*q*>FR.TS == 0b00)

The ERR<*n*>MISC3 (ERR<*q*>FR.TS == 0b00) bit assignments are:

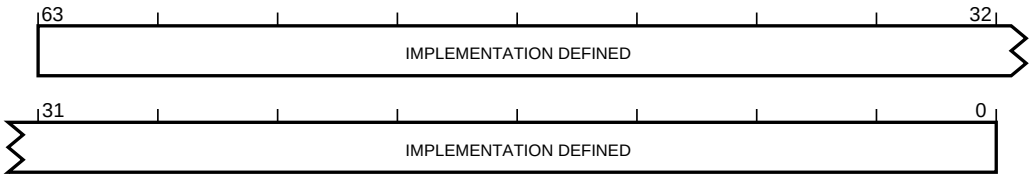

Figure 4.13: ERR<*n*>MISC3

Bits [63:0]

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

4.3.8.3 Accessibility

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 ERR<q>PFGF.MV is 0b1, then some parts of this register are read/write when ERR<n>STATUS.MV == 0b1. 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 == 0b1, 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.

4.3.9 ERR<*n*>PFGCDN, Pseudo-fault Generation Countdown Register

The ERR<*n*>PFGCDN characteristics are:

Purpose

Generates one of the errors enabled in the corresponding ERR<n>PFGCTL register.

Configurations

ERR<*n*>PFGCDN is present only if all of the following are true:

- Error record <*n*> is implemented.
- The node implements the RAS Common Fault Injection Model Extension (ERR<n>FR.INJ != `0b00`).
- Error record <*n*> is the first error record owned by a node.

ERR<*n*>PFGCDN is RES0 otherwise.

ERR<n>FR describes the features implemented by the node.

Attributes

When accessed using a System register, ERR<*n*>PFGCDN is a 64-bit read/write register accessed using `MRS` and `MSR` of ERXPFGCDN_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>PFGCDN is a 64-bit read/write register located at offset `0x810` + 64×*n*.

4.3.9.1 Field descriptions

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

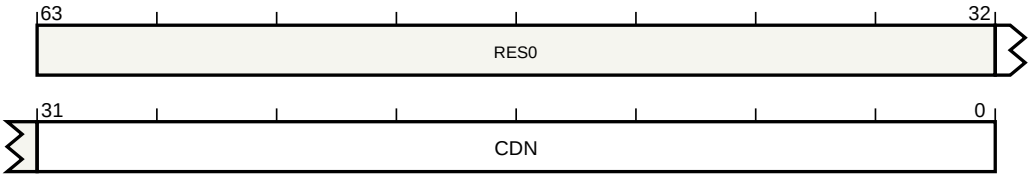

Figure 4.14: ERR<*n*>PFGCDN

Bits [63:32]

Reserved. This field is RES0.

CDN, bits [31:0]

Countdown value.

This field is copied to Error Generation Counter when either:

- Software writes ERR<n>PFGCTL.CDNEN with 1.
- The Error Generation Counter decrements to zero and ERR<n>PFGCTL.R == `0b1`.

While ERR<n>PFGCTL.CDNEN == `0b1` 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 0, one of the errors enabled in the ERR<n>PFGCTL register is generated.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

Note:

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

4.3.9.2 Accessibility

None.

4.3.10 ERR<*n*>PFGCTL, Pseudo-fault Generation Control Register

The ERR<*n*>PFGCTL characteristics are:

Purpose

Enables controlled fault generation.

Configurations

ERR<*n*>PFGCTL is present only if all of the following are true:

- Error record <*n*> is implemented.
- The node implements the RAS Common Fault Injection Model Extension (ERR<n>FR.INJ != `0b00`).
- Error record <*n*> is the first error record owned by a node.

ERR<*n*>PFGCTL is RES0 otherwise.

ERR<n>FR describes the features implemented by the node.

Attributes

When accessed using a System register, ERR<*n*>PFGCTL is a 64-bit read/write register accessed using `MRS` and `MSR` of ERXPFGCTL_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>PFGCTL is a 64-bit read/write register located at offset `0x808` + 64×*n*.

4.3.10.1 Field descriptions

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

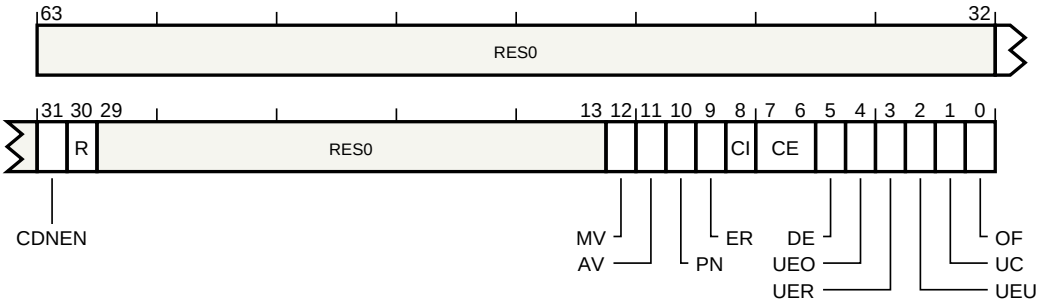

Figure 4.15: ERR<*n*>PFGCTL

Bits [63:32,29:13]

Reserved. This field is RES0.

CDNEN, bit [31]

Countdown Enable. Controls transfers from the value that is held in the ERR<n>PFGCDN into the Error Generation Counter and enables this counter. The possible values of this bit are:

`0`	The Error Generation Counter is disabled.
`1`	The Error Generation Counter is enabled. On a write of `0b1` to this bit, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

R, bit [30]

Restart.

When the node supports this control

Controls whether, upon reaching zero, the Error Generation Counter restarts from the ERR<n>PFGCDN value or stops. The possible values of this bit are:

0	On reaching 0, the Error Generation Counter will stop.
1	On reaching 0, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

MV, bit [12]

Miscellaneous syndrome.

When the node supports this control

The value that is written to ERR<n>STATUS.MV when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.MV is set to 0b0 when an injected error is recorded.
1	ERR<n>STATUS.MV is set to 0b1 when an injected error is recorded.

Accessing this bit has the following behavior:

- This bit reads-as-one and ignores writes if the node always records some syndrome in ERR<*n*>MISC<*m*>, setting ERR<n>STATUS.MV to 1, when an injected error is recorded.
- Otherwise, this bit is read/write.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

AV, bit [11]

Address syndrome.

When the node supports this control

The value that is written to ERR<n>STATUS.AV when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.AV is set to 0b0 when an injected error is recorded.
1	ERR<n>STATUS.AV is set to 0b1 when an injected error is recorded.

Accessing this bit has the following behavior:

- This bit reads-as-one and ignores writes if the node always sets ERR<n>STATUS.AV to 0b1 when an injected error is recorded.
- Otherwise, this bit is read/write.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

PN, bit [10]
Poison flag.

When the node supports this control
The value that is written to ERR<n>STATUS.PN when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.PN is set to 0b0 when an injected error is recorded.
1	ERR<n>STATUS.PN is set to 0b1 when an injected error is recorded.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

ER, bit [9]
Error Reported flag.

When the node supports this control
The value that is written to ERR<n>STATUS.ER when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.ER is set to 0b0 when an injected error is recorded.
1	ERR<n>STATUS.ER is set to 0b1 when an injected error is recorded.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

CI, bit [8]
Critical Error flag.

When the node supports this control
The value that is written to ERR<n>STATUS.CI when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.CI is set to 0b0 when an injected error is recorded.
1	ERR<n>STATUS.CI is set to 0b1 when an injected error is recorded.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

CE, bits [7:6]
Corrected Error generation enable.

When the node supports this control
Controls the type of Corrected Error condition that might be generated. The possible values of this field are:

0b00	No error of this type will be generated.

0b01	A non-specific Corrected Error, that is, a Corrected Error that is recorded as ERR<n>STATUS.CE == 0b10, might be generated when the Error Generation Counter decrements to zero.
0b10	A transient Corrected Error, that is, a Corrected Error that is recorded as ERR<n>STATUS.CE == 0b01, might be generated when the Error Generation Counter decrements to zero.
0b11	A persistent Corrected Error, that is, a Corrected Error that is recorded as ERR<n>STATUS.CE == 0b11, might be generated when the Error Generation Counter decrements to zero.

The set of permitted values for this field is defined by ERR<n>PFGF.CE.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

Otherwise
Reserved. This field is RES0.

DE, bit [5]
Deferred Error generation enable.

When the node supports this control
Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed. The possible values of this bit are:

0	No error of this type will be generated.
1	An error of this type might be generated when the Error Generation Counter decrements to zero.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

UEO, bit [4]
Latent or Restartable Error generation enable.

When the node supports this control
Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed. The possible values of this bit are:

0	No error of this type will be generated.
1	An error of this type might be generated when the Error Generation Counter decrements to zero.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

UER, bit [3]
Signaled or Recoverable Error generation enable.

When the node supports this control

Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed. The possible values of this bit are:

0	No error of this type will be generated.
1	An error of this type might be generated when the Error Generation Counter decrements to zero.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

UEU, bit [2]

Unrecoverable Error generation enable.

When the node supports this control

Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed. The possible values of this bit are:

0	No error of this type will be generated.
1	An error of this type might be generated when the Error Generation Counter decrements to zero.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

UC, bit [1]

Uncontainable Error generation enable.

When the node supports this control

Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed. The possible values of this bit are:

0	No error of this type will be generated.
1	An error of this type might be generated when the Error Generation Counter decrements to zero.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise

Reserved. This bit is RES0.

OF, bit [0]

Overflow flag.

When the node supports this control

The value that is written to ERR<n>STATUS.OF when an injected error is recorded. The possible values of this bit are:

0	ERR<n>STATUS.OF is set to 0b0 when an injected error is recorded.

1	ERR<n>STATUS.OF is set to 0b1 when an injected error is recorded.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

4.3.10.2 Accessibility

None.

4.3.11 ERR<*n*>PFGF, Pseudo-fault Generation Feature Register

The ERR<*n*>PFGF characteristics are:

Purpose

Defines which common architecturally-defined fault generation features are implemented.

Configurations

ERR<*n*>PFGF is present only if all of the following are true:

- Error record <*n*> is implemented.
- The node implements the RAS Common Fault Injection Model Extension (ERR<n>FR.INJ != `0b00`).
- Error record <*n*> is the first error record owned by a node.

ERR<*n*>PFGF is RES0 otherwise.

ERR<n>FR describes the features implemented by the node.

Attributes

When accessed using a System register, ERR<*n*>PFGF is a 64-bit read-only register accessed using `MRS` of ERXPFGF_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>PFGF is a 64-bit read-only register located at offset `0x800` + 64×*n*.

4.3.11.1 Field descriptions

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

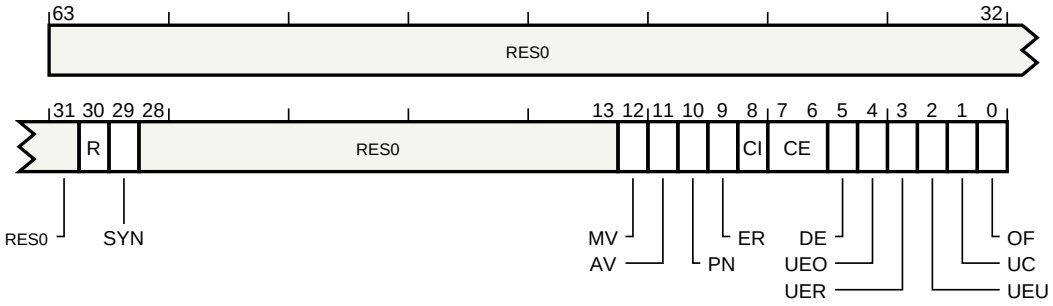

Figure 4.16: ERR<*n*>PFGF

Bits [63:31,28:13]

Reserved. This field is RES0.

R, bit [30]

Restartable. Support for Error Generation Counter restart mode. The defined values of this bit are:

`0`	The node does not support this feature.
`1`	Feature controllable.

SYN, bit [29]

Syndrome. Fault syndrome injection. The defined values of this bit are:

`0`	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 == `0b0`.

1	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 == 0b0.

Note:

If ERR<*n*>PFGF.SYN == 0b1, 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.

MV, bit [12]

Miscellaneous syndrome.

Additional syndrome injection. 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 syndrome fields in ERR<*n*>MISC<*m*> this refers to, as some fields might always be recorded by an error. For example, a Corrected Error counter.

The defined values of this bit are:

0	When an injected error is recorded, the node might record IMPLEMENTATION DEFINED additional syndrome in ERR<*n*>MISC<*m*>. If any syndrome is recorded in ERR<*n*>MISC<*m*>, then ERR<n>STATUS.MV is set to 0b1.
1	When an injected error is recorded, the node does not update all the syndrome fields in the ERR<*n*>MISC<*m*> and does one of: • The node does not update any fields in ERR<*n*>MISC<*m*> and sets ERR<n>STATUS.MV to ERR<n>PFGCTL.MV. • The node records some syndrome in ERR<*n*>MISC<*m*> and sets ERR<n>STATUS.MV to 0b1. ERR<n>PFGCTL.MV is RAO/WI. The syndrome fields that the node does not update are unchanged and are writable when ERR<n>STATUS.MV == 0b0.

Note:

If ERR<*n*>PFGF.MV == 0b1, software can write specific values into the ERR<*n*>MISC<*m*> registers when setting up a fault injection event. The values that can be written to these registers are IMPLEMENTATION DEFINED.

AV, bit [11]

Address syndrome. Address syndrome injection. The defined values of this bit are:

0	When an injected error is recorded, the node either sets ERR<n>ADDR and ERR<n>STATUS.AV for the access, or leaves these unchanged.
1	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. • Sets ERR<n>STATUS.AV to 0b1. ERR<n>PFGCTL.AV is RAO/WI. ERR<n>ADDR is writable when ERR<n>STATUS.AV == 0b0.

Note:

If ERR<*n*>PFGF.AV == 0b1, software can write a specific value into ERR<n>ADDR when setting up a fault injection event.

PN, bit [10]
Poison flag.

When the node supports this flag
Describes how the fault generation feature of the node sets the ERR<n>STATUS.PN status flag. The defined values of this bit are:

0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.PN to 0b1.
1	When an injected error is recorded, ERR<n>STATUS.PN is set to ERR<n>PFGCTL.PN.

This behavior replaces the architecture-defined rules for setting the PN bit.

Otherwise
This bit reads-as-zero.

ER, bit [9]
Error Reported flag.

When the node supports this flag
Describes how the fault generation feature of the node sets the ERR<n>STATUS.ER status flag. The defined values of this bit are:

0	When an injected error is recorded, the node sets ERR<n>STATUS.ER according to the architecture-defined rules for setting the ER bit.
1	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.

Otherwise
This bit reads-as-zero.

CI, bit [8]
Critical Error flag.

When the node supports this flag
Describes how the fault generation feature of the node sets the ERR<n>STATUS.CI status flag. The defined values of this bit are:

0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.CI to 0b1.
1	When an injected error is recorded, ERR<n>STATUS.CI is set to ERR<n>PFGCTL.CI.

This behavior replaces the architecture-defined rules for setting the CI bit.

Otherwise
This bit reads-as-zero.

CE, bits [7:6]
Corrected Error generation.

When the node supports this type of error
Describes the types of Corrected Error that the fault generation feature of the node can generate. The defined values of this field are:

0b00	The fault generation feature of the node cannot generate this type of error.
0b01	The fault generation feature of the node allows generation of a non-specific Corrected Error, that is, a Corrected Error that is recorded as ERR<n>STATUS.CE == 0b10.
0b11	The fault generation feature of the node allows generation of transient or persistent Corrected Errors, that is, Corrected Errors that are recorded as ERR<n>STATUS.CE == 0b01 and 0b11.

All other values are reserved.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.CE indicates whether the node supports this type of error.

Otherwise
This field reads-as-zero.

DE, bit [5]
Deferred Error generation.

When the node supports this type of error
Describes whether the fault generation feature of the node can generate this type of error. The defined values of this bit are:

0	The fault generation feature of the node cannot generate this type of error.
1	The fault generation feature of the node allows generation of this type of error.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.DE indicates whether the node supports this type of error.

Otherwise
This bit reads-as-zero.

UEO, bit [4]
Latent or Restartable Error generation.

When the node supports this type of error
Describes whether the fault generation feature of the node can generate this type of error. The defined values of this bit are:

0	The fault generation feature of the node cannot generate this type of error.
1	The fault generation feature of the node allows generation of this type of error.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.UEO indicates whether the node supports this type of error.

Otherwise
This bit reads-as-zero.

UER, bit [3]
Signaled or Recoverable Error generation.

When the node supports this type of error
Describes whether the fault generation feature of the node can generate this type of error. The defined values of this bit are:

0	The fault generation feature of the node cannot generate this type of error.

1	The fault generation feature of the node allows generation of this type of error.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.UER indicates whether the node supports this type of error.

Otherwise
This bit reads-as-zero.

UEU, bit [2]
Unrecoverable Error generation.

When the node supports this type of error
Describes whether the fault generation feature of the node can generate this type of error. The defined values of this bit are:

0	The fault generation feature of the node cannot generate this type of error.
1	The fault generation feature of the node allows generation of this type of error.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.UEU indicates whether the node supports this type of error.

Otherwise
This bit reads-as-zero.

UC, bit [1]
Uncontainable Error generation.

When the node supports this type of error
Describes whether the fault generation feature of the node can generate this type of error. The defined values of this bit are:

0	The fault generation feature of the node cannot generate this type of error.
1	The fault generation feature of the node allows generation of this type of error.

If ERR<n>FR.FRX is 0b1 then ERR<n>FR.UC indicates whether the node supports this type of error.

Otherwise
This bit reads-as-zero.

OF, bit [0]
Overflow flag.

When the node supports this flag
Describes how the fault generation feature of the node sets the ERR<n>STATUS.OF status flag. The defined values of this bit are:

0	When an injected error is recorded, the node sets ERR<n>STATUS.OF according to the architecture-defined rules for setting the OF bit.
1	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.

Otherwise
This bit reads-as-zero.

4.3.11.2 Accessibility

None.

4.3.12 ERR<*n*>STATUS, Error Record Primary Status Register

The ERR<*n*>STATUS characteristics are:

Purpose

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:

- The {AV, V, MV} bits are valid bits that define whether error record <*n*> registers are valid.
- The {UE, OF, CE, DE, UET} bits encode the types of error or errors recorded.
- The {CI, ER, PN, IERR, SERR} fields are syndrome fields.

Configurations

ERR<*n*>STATUS is present only if error record <*n*> is implemented. ERR<*n*>STATUS is RES0 otherwise.

ERR<q>FR describes the features implemented by the node that owns error record <*n*>. <*q*> 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 $q = n$.

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, non-zero, 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 ERR<q>CTLR.

Attributes

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

- `MRC` and `MCR` of ERXSTATUS for ERR<*n*>STATUS[31:0] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXSTATUS_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as a memory-mapped register, ERR<*n*>STATUS is a 64-bit read/write register located at offset `0x010` + 64×*n*.

4.3.12.1 ERR<*n*>STATUS (RAS System Architecture v1.1 is implemented)

The ERR<*n*>STATUS (RAS System Architecture v1.1 is implemented) bit assignments are:

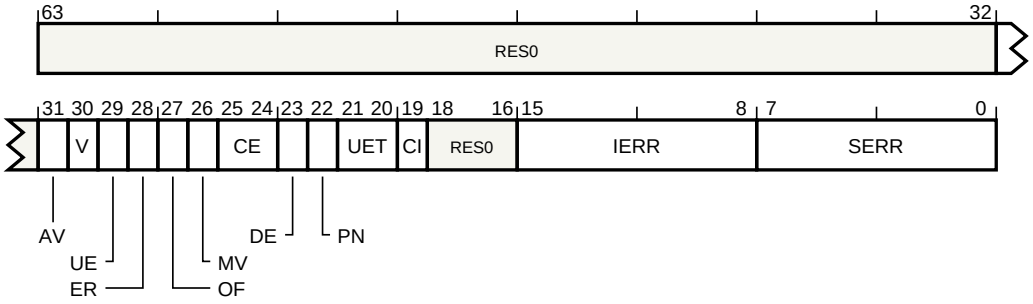

Figure 4.17: ERR<*n*>STATUS

Bits [63:32,18:16]
Reserved. This field is RES0.

AV, bit [31]
Address Valid.

When error record <*n*> includes an address associated with an error
The possible values of this bit are:

0	ERR<n>ADDR not valid.
1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

V, bit [30]
Status Register Valid. The possible values of this bit are:

0	ERR<*n*>STATUS not valid.
1	ERR<*n*>STATUS valid. At least one error has been recorded.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

UE, bit [29]
Uncorrected Error. The possible values of this bit are:

0	No errors have been detected, or all detected errors have been either corrected or deferred.
1	At least one detected error was not corrected and not deferred.

When clearing ERR<*n*>STATUS.V to `0b0`, if this bit is nonzero, then Arm recommends that software write `0b1` to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

ER, bit [28]
Error Reported. The possible values of this bit are:

0	No in-band error (External Abort) reported.
1	An External Abort 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 applicable one of the ERR<q>CTLR.{WUE,RUE,UE} bits is implemented and was set to 0b1 when an Uncorrected error was detected. • The applicable one of the ERR<q>CTLR.{WUE,RUE,UE} bits is not implemented and the component always reports errors.

It is IMPLEMENTATION DEFINED whether this bit can be set to 0b1 by a Deferred error.

When clearing ERR<*n*>STATUS.V to 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - This bit is never set to 0b1 by a Deferred error.
  - All of the following are true:
    - ERR<*n*>STATUS.{UE,DE} == {0,0}.
    - This bit can be set to 0b1 by a Deferred error.
  - ERR<*n*>STATUS.V == 0b0.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Note:

An External Abort signaled by the component might be masked and not generate any exception.

OF, bit [27]
Overflow.

Indicates that multiple errors have been detected. This bit is set to 0b1 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 set to 0b1, a Corrected error counter is not implemented, and a Corrected error is recorded.
- ERR<*n*>STATUS.V was previously set to 0b1, and a type of error other than a Corrected error is recorded.

Otherwise, this bit is unchanged when an error is recorded.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.
- A direct write to this bit that clears this bit to zero might indirectly set the counter overflow flag to an UNKNOWN value.

The possible values of this bit are:

0	Since this bit was last cleared to zero, no error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed.
1	Since this bit 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 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

MV, bit [26]
Miscellaneous Registers Valid.

When error record <*n*> includes an additional information for an error
The possible values of this bit are:

0	ERR<*n*>MISC<*m*> not valid.
1	The IMPLEMENTATION DEFINED contents of the ERR<*n*>MISC<*m*> registers contains additional information for an error recorded by this record.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

Otherwise
Reserved. This bit is RES0.

CE, bits [25:24]
Corrected Error. The possible values of this field are:

0b00	No errors were corrected.
0b01	At least one transient error was corrected.
0b10	At least one error was corrected.
0b11	At least one persistent error was corrected.

The mechanism by which a component or node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<*n*>STATUS.V to 0b0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- Otherwise, this field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

DE, bit [23]
Deferred Error. The possible values of this bit are:

0	No errors were deferred.
1	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<*n*>STATUS.V to 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

PN, bit [22]
Poison. The possible values of this bit are:

0	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.
1	Uncorrected error or Deferred error recorded because a poison value was detected.

When clearing ERR<*n*>STATUS.V to 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if any of the following are true:
  - ERR<*n*>STATUS.V == 0b0.
  - ERR<*n*>STATUS.{DE,UE} == {0,0}.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

UET, bits [21:20]
Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error. The possible values of this field are:

0b00	Uncorrected error, Uncontainable error (UC).
0b01	Uncorrected error, Unrecoverable error (UEU).
0b10	Uncorrected error, Latent or Restartable error (UEO).
0b11	Uncorrected error, Signaled or Recoverable error (UER).

When clearing ERR<*n*>STATUS.V to `0b0`, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if any of the following are true:
  - ERR<*n*>STATUS.V == `0b0`.
  - ERR<*n*>STATUS.UE == `0b0`.
- Otherwise, this field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

Note:

Software might use the information in the error record registers to determine what recovery is necessary.

CI, bit [19]

Critical Error. Indicates whether a critical error condition has been recorded. The possible values of this bit are:

`0`	No critical error condition.
`1`	Critical error condition.

When clearing ERR<*n*>STATUS.V to `0b0`, if this bit is nonzero, then Arm recommends that software write `0b1` to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == `0b0`.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if all of the following are true:
  - Any of the following are true:
    - The RAS Common Fault Injection Model Extension is not implemented by the node that owns this error record.
    - ERR<q>PFGF.SYN == `0b0`.
  - ERR<*n*>STATUS.V == `0b0`.
- Otherwise, this field is read/write.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

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. The possible values of this field are:

0	No error.
1	IMPLEMENTATION DEFINED error.
2	Data value from (non-associative) internal memory. For example, *Error Correction Code* (ECC) from on-chip SRAM or buffer.
3	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
4	Assertion failure. For example, consistency failure.
5	Error detected on internal data path. For example, parity on ALU result.
6	Data value from associative memory. For example, ECC error on cache data.
7	Address/control value from associative memory. For example, ECC error on cache tag.
8	Data value from a TLB. For example, ECC error on TLB data.
9	Address/control value from a TLB. For example, ECC error on TLB tag.
10	Data value from producer. For example, parity error on write data bus.
11	Address/control value from producer. For example, parity error on address bus.
12	Data value from (non-associative) external memory. For example, ECC error in SDRAM.
13	Illegal address (software fault). For example, access to unpopulated memory.
14	Illegal access (software fault). For example, byte write to word register.
15	Illegal state (software fault). For example, device not ready.
16	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, and SVE registers are data registers.
17	Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, and SVE registers are control registers.
18	Error response from Completer of access. For example, error response from cache write-back.
19	External timeout. For example, timeout on interaction with another component.
20	Internal timeout. For example, timeout on interface within the component.
21	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.
22	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.
23	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
24	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
25	Error recorded by *Peripheral Component Interconnect Express* (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.
26	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.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if all of the following are true:
  - Any of the following are true:
    - The RAS Common Fault Injection Model Extension is not implemented by the node that owns this error record.
    - ERR<q>PFGF.SYN == 0b0.
  - ERR<*n*>STATUS.V == 0b0.
- Otherwise, this field is read/write.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

4.3.12.2 ERR<*n*>STATUS (RAS System Architecture v1.0 is implemented)

The ERR<*n*>STATUS (RAS System Architecture v1.0 is implemented) bit assignments are:

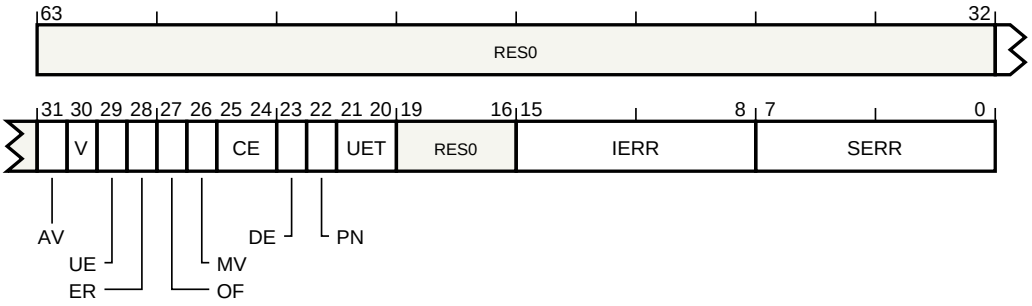

Figure 4.18: ERR<*n*>STATUS

Bits [63:32,19:16]
Reserved. This field is RES0.

AV, bit [31]
Address Valid.

When error record <*n*> includes an address associated with an error
The possible values of this bit are:

0	ERR<n>ADDR not valid.
1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

Accessing this bit has the following behavior:

- This bit ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.

        - ERR<*n*>STATUS.DE is not being cleared to `0b0` in the same write.
      - All of the following are true:
        - ERR<*n*>STATUS.{DE,UE} == {0,0}.
        - ERR<*n*>STATUS.CE != `0b00`.
        - ERR<*n*>STATUS.CE is not being cleared to `0b00` in the same write.
    - Otherwise, this bit is read/write-one-to-clear.

    This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

Otherwise
Reserved. This bit is RES0.

V, bit [30]
Status Register Valid. The possible values of this bit are:

`0`	ERR<*n*>STATUS not valid.
`1`	ERR<*n*>STATUS valid. At least one error has been recorded.

Accessing this bit has the following behavior:

- This bit ignores writes if any of the following are true:
  - 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.
  - 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.
  - All of the following are true:
    - ERR<*n*>STATUS.CE != `0b00`.
    - ERR<*n*>STATUS.CE is not being cleared to `0b00` in the same write.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

UE, bit [29]
Uncorrected Error. The possible values of this bit are:

`0`	No errors have been detected, or all detected errors have been either corrected or deferred.
`1`	At least one detected error was not corrected and not deferred.

When clearing ERR<*n*>STATUS.V to `0b0`, if this bit is nonzero, then Arm recommends that software write `0b1` to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == `0b0`.
- This bit ignores writes if all of the following are true:
  - ERR<*n*>STATUS.OF == `0b1`.
  - ERR<*n*>STATUS.OF is not being cleared to `0b0` in the same write.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

ER, bit [28]
Error Reported. The possible values of this bit are:

0	No in-band error (External Abort) reported.
1	An External Abort 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 applicable one of the ERR<q>CTLR.{WUE,RUE,UE} bits is implemented and was set to 0b1 when an Uncorrected error was detected. • The applicable one of the ERR<q>CTLR.{WUE,RUE,UE} bits is not implemented and the component always reports errors.

It is IMPLEMENTATION DEFINED whether this bit can be set to 0b1 by a Deferred error.

If this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0b0.
- Clearing ERR<*n*>STATUS.UE to 0b0, if this bit is never set to 0b1 by a Deferred error.
- Clearing ERR<*n*>STATUS.{UE,DE} to {0,0}, if this bit can be set to 0b1 by a Deferred error.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - This bit is never set to 0b1 by a Deferred error.
  - All of the following are true:
    - ERR<*n*>STATUS.{UE,DE} == {0,0}.
    - This bit can be set to 0b1 by a Deferred error.
  - ERR<*n*>STATUS.V == 0b0.
- This bit ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.{DE,UE} == {0,0}.
    - ERR<*n*>STATUS.CE != 0b00.
    - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

Note:

An External Abort signaled by the component might be masked and not generate any exception.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This bit is set to 0b1 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 0b1.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0b0 and ERR<*n*>STATUS.DE == 0b1.

- A Corrected error is detected, no Corrected error counter is implemented, ERR<*n*>STATUS.UE == 0b0, ERR<*n*>STATUS.DE == 0b0, 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 == 0b0, ERR<*n*>STATUS.DE == 0b0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is set to 0b1 when one of the following occurs:

- A Deferred error is detected and ERR<*n*>STATUS.UE == 0b1.
- A Corrected error is detected, no Corrected error counter is implemented, and either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 0b1.
- A Corrected error counter is implemented, either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 0b1, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is cleared to 0b0 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 0b0.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0b0 and ERR<*n*>STATUS.DE == 0b0.
- A Corrected error is detected, ERR<*n*>STATUS.UE == 0b0, ERR<*n*>STATUS.DE == 0b0 and ERR<*n*>STATUS.CE == 0b00.

The IMPLEMENTATION DEFINED clearing of this bit might also depend on the value of the other error status bits.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.
- A direct write to this bit that clears this bit to 0b0 might indirectly set the counter overflow flag to an UNKNOWN value.

The possible values of this bit are:

0	If ERR<*n*>STATUS.UE == 0b1, then no error syndrome for an Uncorrected error has been discarded. If ERR<*n*>STATUS.UE == 0b0 and ERR<*n*>STATUS.DE == 0b1, then no error syndrome for a Deferred error has been discarded. If ERR<*n*>STATUS.UE == 0b0, ERR<*n*>STATUS.DE == 0b0, and a Corrected error counter is implemented, then the counter has not overflowed. If ERR<*n*>STATUS.UE == 0b0, ERR<*n*>STATUS.DE == 0b0, ERR<*n*>STATUS.CE != 0b00, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded. **Note:** This bit might have been set to 0b1 when an error syndrome was discarded and later cleared to 0b0 when a higher priority syndrome was recorded.
1	At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

When clearing ERR<*n*>STATUS.V to 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

MV, bit [26]

Miscellaneous Registers Valid.

When error record <*n*> includes an additional information for an error

The possible values of this bit are:

0	ERR<*n*>MISC<*m*> not valid.
1	The IMPLEMENTATION DEFINED contents of the ERR<*n*>MISC<*m*> registers contains additional information for an error recorded by this record.

Accessing this bit has the following behavior:

- This bit ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.{DE,UE} == {0,0}.
    - ERR<*n*>STATUS.CE != 0b00.
    - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

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.

Otherwise

Reserved. This bit is RES0.

CE, bits [25:24]

Corrected Error. The possible values of this field are:

0b00	No errors were corrected.
0b01	At least one transient error was corrected.
0b10	At least one error was corrected.
0b11	At least one persistent error was corrected.

The mechanism by which a component or node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<*n*>STATUS.V to 0b0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- This field ignores writes if all of the following are true:

  - ERR<*n*>STATUS.OF == 0b1.
  - ERR<*n*>STATUS.OF is not being cleared to 0b0 in the same write.
- Otherwise, this field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

DE, bit [23]
Deferred Error. The possible values of this bit are:

0	No errors were deferred.
1	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<*n*>STATUS.V to 0b0, if this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V == 0b0.
- This bit ignores writes if all of the following are true:
  - ERR<*n*>STATUS.OF == 0b1.
  - ERR<*n*>STATUS.OF is not being cleared to 0b0 in the same write.
- Otherwise, this bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

PN, bit [22]
Poison. The possible values of this bit are:

0	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.
1	Uncorrected error or Deferred error recorded because a poison value was detected.

If this bit is nonzero, then Arm recommends that software write 0b1 to this bit to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0b0.
- Clearing both ERR<*n*>STATUS.{DE, UE} to 0b0.

Accessing this bit has the following behavior:

- This bit is not valid and reads UNKNOWN if any of the following are true:
  - ERR<*n*>STATUS.V == 0b0.
  - ERR<*n*>STATUS.{DE,UE} == {0,0}.
- This bit ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:

        - ERR<*n*>STATUS.{DE,UE} == {0,0}.
        - ERR<*n*>STATUS.CE != 0b00.
        - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
    - Otherwise, this bit is read/write-one-to-clear.

    This bit resets to an architecturally UNKNOWN value on a Cold reset. This bit is preserved on an Error Recovery reset.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error. The possible values of this field are:

0b00	Uncorrected error, Uncontainable error (UC).
0b01	Uncorrected error, Unrecoverable error (UEU).
0b10	Uncorrected error, Latent or Restartable error (UEO).
0b11	Uncorrected error, Signaled or Recoverable error (UER).

If this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0b0.
- Clearing ERR<*n*>STATUS.UE to 0b0.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if any of the following are true:
  - ERR<*n*>STATUS.V == 0b0.
  - ERR<*n*>STATUS.UE == 0b0.
- This field ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.{DE,UE} == {0,0}.
    - ERR<*n*>STATUS.CE != 0b00.
    - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
- Otherwise, this field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

Note:

Software might use the information in the error record registers to determine what recovery is necessary.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code 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.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if all of the following are true:
  - Any of the following are true:
    - The RAS Common Fault Injection Model Extension is not implemented by the node that owns this error record.
    - ERR<q>PFGF.SYN == 0b0.
  - ERR<*n*>STATUS.V == 0b0.
- This field ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.{DE,UE} == {0,0}.
    - ERR<*n*>STATUS.CE != 0b00.
    - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
- Otherwise, this field is read/write.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

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. The possible values of this field are:

0	No error.
1	IMPLEMENTATION DEFINED error.
2	Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.
3	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
4	Assertion failure. For example, consistency failure.
5	Error detected on internal data path. For example, parity on ALU result.
6	Data value from associative memory. For example, ECC error on cache data.
7	Address/control value from associative memory. For example, ECC error on cache tag.
8	Data value from a TLB. For example, ECC error on TLB data.
9	Address/control value from a TLB. For example, ECC error on TLB tag.
10	Data value from producer. For example, parity error on write data bus.
11	Address/control value from producer. For example, parity error on address bus.
12	Data value from (non-associative) external memory. For example, ECC error in SDRAM.
13	Illegal address (software fault). For example, access to unpopulated memory.
14	Illegal access (software fault). For example, byte write to word register.
15	Illegal state (software fault). For example, device not ready.
16	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, and SVE registers are data registers.

17	Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, and SVE registers are control registers.
18	Error response from Completer of access. For example, error response from cache write-back.
19	External timeout. For example, timeout on interaction with another component.
20	Internal timeout. For example, timeout on interface within the component.
21	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.
22	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.
23	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
24	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
25	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.
26	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.

Accessing this field has the following behavior:

- This field is not valid and reads UNKNOWN if all of the following are true:
  - Any of the following are true:
    - The RAS Common Fault Injection Model Extension is not implemented by the node that owns this error record.
    - ERR<q>PFGF.SYN == 0b0.
  - ERR<*n*>STATUS.V == 0b0.
- This field ignores writes if any of the following are true:
  - All of the following are true:
    - ERR<*n*>STATUS.UE != 0b0.
    - ERR<*n*>STATUS.UE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.UE == 0b0.
    - ERR<*n*>STATUS.DE != 0b0.
    - ERR<*n*>STATUS.DE is not being cleared to 0b0 in the same write.
  - All of the following are true:
    - ERR<*n*>STATUS.{DE,UE} == {0,0}.
    - ERR<*n*>STATUS.CE != 0b00.
    - ERR<*n*>STATUS.CE is not being cleared to 0b00 in the same write.
- Otherwise, this field is read/write.

This field resets to an architecturally UNKNOWN value on a Cold reset. This field is preserved on an Error Recovery reset.

4.3.12.3 Accessibility

The {AV, V, UE, ER, OF, MV, CE, DE, PN, UET, CI} fields are write-one-to-clear, meaning writes of zero are ignored, and a write of one or all-ones to the field clears the field to zero. The {IERR, SERR} fields are read/write fields, although the set of implemented valid values is IMPLEMENTATION DEFINED. See also ERR<n>PFGF.SYN.

After reading ERR<*n*>STATUS, software must clear the valid bits in the register to allow new errors to be recorded. However, between reading the register and clearing the valid bits, 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 the {UE, DE, CE} fields are ignored if the OF bit is set and is not being cleared.
- Writes to the V bit are ignored if any of the {UE, DE, CE} fields are nonzero and are not being cleared.
- Writes to the {AV, MV} bits and {ER, PN, UET, IERR, SERR} syndrome fields are ignored if the highest priority nonzero error status field is nonzero is not being cleared. The error status fields in priority order from highest to lowest, are UE, DE, and CE.

When RAS System Architecture v1.1 is implemented, a write to the register is ignored if all of:

- Any of {V, UE, OF, CE, DE} fields are nonzero before the write.
- The write does not clear the nonzero {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 the {V, DE, UE} status fields 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 {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 bits in the register to allow new errors to be recorded, Arm recommends that software:

- Determine which fields to clear to zero by reading ERR<*n*>STATUS.
- Write ones to all the write-one-to-clear fields that are nonzero.
- Write zero to all the read/write fields.
- Write zero to all the write-one-to-clear fields that are zero.

Otherwise, these fields might not have the correct value when a new fault is recorded.

An exception is when the node supports writing to these fields as part of fault injection. See also ERR<n>PFGF.SYN.

ERR<*n*>STATUS ignores writes if all of the following are true:

- Any of the following are true:
  - ERR<*n*>STATUS.V != `0b0` and ERR<*n*>STATUS.V is not being cleared to `0b0` in the same write.
  - ERR<*n*>STATUS.UE != `0b0` and ERR<*n*>STATUS.UE is not being cleared to `0b0` in the same write.
  - ERR<*n*>STATUS.OF != `0b0` and ERR<*n*>STATUS.OF is not being cleared to `0b0` in the same write.
  - ERR<*n*>STATUS.CE != `0b00` and ERR<*n*>STATUS.CE is not being cleared to `0b0` in the same write.
  - ERR<*n*>STATUS.DE != `0b0` and ERR<*n*>STATUS.DE is not being cleared to `0b0` in the same write.
- RAS System Architecture v1.1 is implemented.

4.3.12.4 Pseudocode operation

```
// ERRSTATUS[] (assignment form)
// =============================
// For a system register, n = UInt(ERRSELR_EL1.SEL)

ERRSTATUS[integer n] = bits(64) w

    // Generate candidate value from the written value and the previous
    // (physical register) value
    c = w<63:32>:(_ERRSTATUS[n]<31:19> AND NOT(w<31:19>)):Zeros(3):w<15:0>;

    if HaveRASSysArchv1p1() then
        // RAS System Architecture v1.1
        // - ignore write if any of V/UE/DE/CE/OF is set
        if !IsZero(c.<V,UE,OF,CE,DE>) then
            c = _ERRSTATUS[n];
    else
        // RAS System Architecture v1.0
        // - do not clear UE/DE/CE if OF is set
        if c.OF == '1' then c.<UE,DE,CE> = _ERRSTATUS[n].<UE,DE,CE>;
        // - do not clear V if any of UE/DE/CE is set
        if !IsZero(c.<UE,DE,CE>) then c.V = _ERRSTATUS[n].V;
        // - do not clear syndrome if not clearing highest priority error
        if (c.UE != '0' ||
            (_ERRSTATUS[n].UE == '0' && c.DE != '0') ||
            (_ERRSTATUS[n].<UE,DE> == '00' && c.CE != '00')) then
            c.<AV,ER,MV,PN,CI,UET,IERR,SERR> = _ERRSTATUS.<AV,ER,MV,PN,CI,UET,IERR,SERR>;

    _ERRSTATUS[n] = c;

    return;
```

4.3.13 ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRCIDR0 is present. ERRCIDR0 is RES0 if not present.

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

Attributes
ERRCIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFF0`.

4.3.13.1 Field descriptions

The ERRCIDR0 bit assignments are:

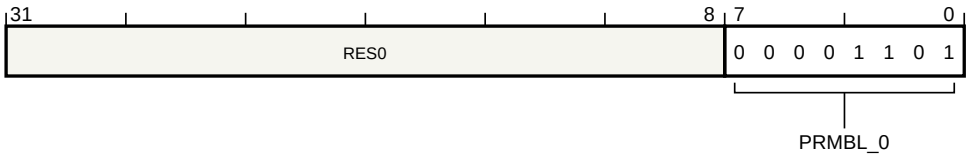

Figure 4.19: ERRCIDR0

Bits [31:8]
Reserved. This field is RES0.

PRMBL_0, bits [7:0]
Component identification preamble, segment 0. This field reads as `0x0D`.

4.3.13.2 Accessibility

None.

4.3.14 ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRCIDR1 is present. ERRCIDR1 is RES0 if not present.

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

Attributes
ERRCIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFF4`.

4.3.14.1 Field descriptions

The ERRCIDR1 bit assignments are:

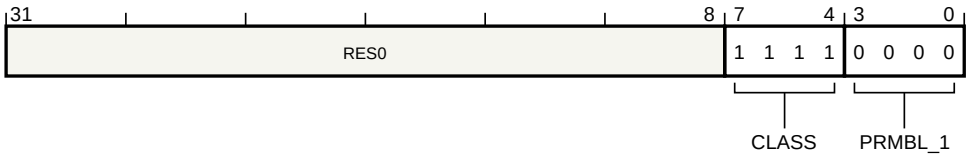

Figure 4.20: ERRCIDR1

Bits [31:8]
Reserved. This field is RES0.

CLASS, bits [7:4]
Component class. The defined values of this field are:

`0xF`	Generic peripheral with IMPLEMENTATION DEFINED register layout.

Other values are defined by the CoreSight Architecture.

This field reads as `0xF`.

PRMBL_1, bits [3:0]
Component identification preamble, segment 1. This field reads as `0x0`.

4.3.14.2 Accessibility

None.

4.3.15 ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRCIDR2 is present. ERRCIDR2 is RES0 if not present.

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

Attributes
ERRCIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFF8`.

4.3.15.1 Field descriptions

The ERRCIDR2 bit assignments are:

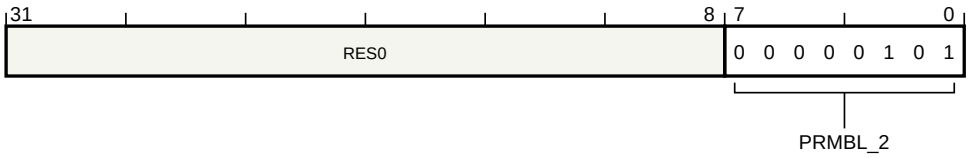

Figure 4.21: ERRCIDR2

Bits [31:8]
Reserved. This field is RES0.

PRMBL_2, bits [7:0]
Component identification preamble, segment 2. This field reads as `0x05`.

4.3.15.2 Accessibility

None.

4.3.16 ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRCIDR3 is present. ERRCIDR3 is RES0 if not present.

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

Attributes
ERRCIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFFC`.

4.3.16.1 Field descriptions

The ERRCIDR3 bit assignments are:

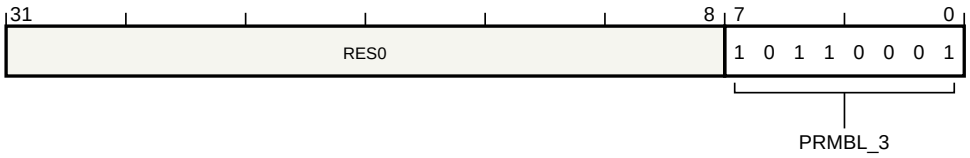

Figure 4.22: ERRCIDR3

Bits [31:8]
Reserved. This field is RES0.

PRMBL_3, bits [7:0]
Component identification preamble, segment 3. This field reads as `0xB1`.

4.3.16.2 Accessibility

None.

4.3.17 ERRCRICR0, Critical Error Interrupt Configuration Register 0

The ERRCRICR0 characteristics are:

Purpose
Critical Error Interrupt configuration register.

Configurations
ERRCRICR0 is present only if all of the following are true:

- Any of the following are true:
  - The Critical Error Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRCRICR0 is RES0 otherwise.

ERRCRICR0 is architecturally mapped to memory-mapped register ERRIRQCR4[63:0].

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

Attributes
ERRCRICR0 is a 64-bit read/write memory-mapped register located at offset `0xEA0`.

4.3.17.1 Critical Error Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Critical Error Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Critical Error Interrupt is implemented, recommended layout bit assignments are:

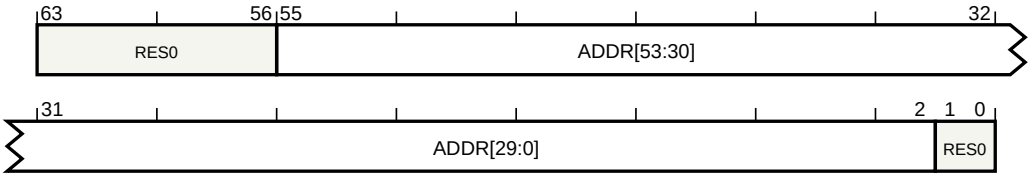

Figure 4.23: ERRCRICR0 Critical Error Interrupt is implemented, recommended layout

Bits [63:56,1:0]
Reserved. This field is 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.

This field resets to an architecturally UNKNOWN value on a reset.

4.3.17.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

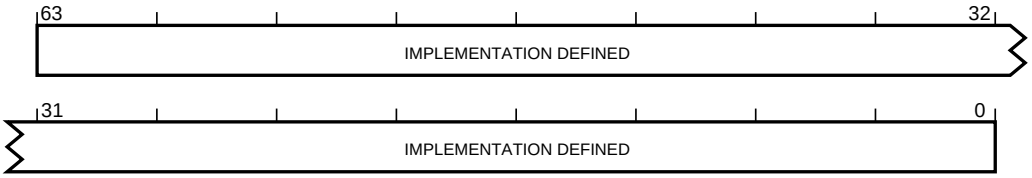

Figure 4.24: ERRCRICR0 IMPLEMENTATION DEFINED layout

Bits [63:0]

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

4.3.17.3 Accessibility

None.

4.3.18 ERRCRICR1, Critical Error Interrupt Configuration Register 1

The ERRCRICR1 characteristics are:

Purpose
Critical Error Interrupt configuration register.

Configurations
ERRCRICR1 is present only if all of the following are true:

- Any of the following are true:
  - The Critical Error Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRCRICR1 is RES0 otherwise.

ERRCRICR1 is architecturally mapped to memory-mapped register ERRIRQCR5[31:0].

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

Attributes
ERRCRICR1 is a 32-bit read/write memory-mapped register located at offset `0xEA8`.

4.3.18.1 Critical Error Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Critical Error Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Critical Error Interrupt is implemented, recommended layout bit assignments are:

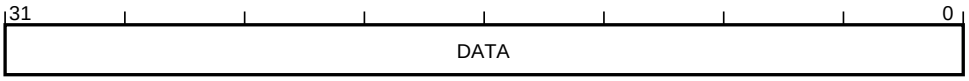

Figure 4.25: ERRCRICR1 Critical Error Interrupt is implemented, recommended layout

DATA, bits [31:0]
Payload for the message signaled interrupt. This field resets to an architecturally UNKNOWN value on a reset.

4.3.18.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

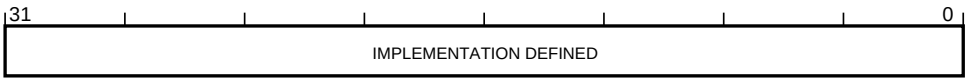

Figure 4.26: ERRCRICR1 IMPLEMENTATION DEFINED layout

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

4.3.18.3 Accessibility

None.

4.3.19 ERRCRICR2, Critical Error Interrupt Configuration Register 2

The ERRCRICR2 characteristics are:

Purpose
Critical Error Interrupt control and configuration register.

Configurations
ERRCRICR2 is present only if all of the following are true:

- Any of the following are true:
  - The Critical Error Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRCRICR2 is RES0 otherwise.

ERRCRICR2 is architecturally mapped to memory-mapped register ERRIRQCR5[63:32].

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

Attributes
ERRCRICR2 is a 32-bit read/write memory-mapped register located at offset `0xEAC`.

4.3.19.1 Critical Error Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Critical Error Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Critical Error Interrupt is implemented, recommended layout bit assignments are:

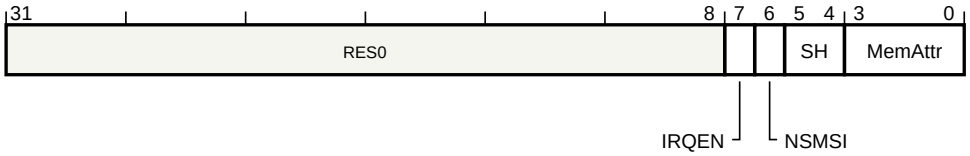

Figure 4.27: ERRCRICR2 Critical Error Interrupt is implemented, recommended layout

Bits [31:8]
Reserved. This field is RES0.

IRQEN, bit [7]
Message signaled interrupt enable.

When the component supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

`0b0`	Disabled.
`0b1`	Enabled.

This bit resets to `0b0` on a reset.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [6]
Security attribute. Defines the physical address space for message signaled interrupts.

When the component supports configuring the Security attribute for message signaled interrupts, and the component does not allow Non-secure writes to ERRCRICR2
The possible values of this bit are:

0b0	Secure.
0b1	Non-secure.

This bit resets to an IMPLEMENTATION DEFINED value on a reset.

When the component allows Non-secure writes to ERRCRICR2
The Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0.

Otherwise
The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [5:4]
Shareability.

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

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.

This field resets to an architecturally UNKNOWN value on a reset.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [3:0]
Memory type.

When the component supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

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

All other values are reserved.

This field resets to an architecturally UNKNOWN value on a reset.

Note:

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

Otherwise

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

This field is RES0.

4.3.19.2 IMPLEMENTATION DEFINED layout

Configurations

Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

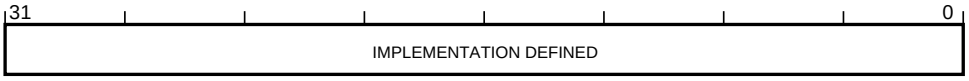

Figure 4.28: ERRCRICR2 IMPLEMENTATION DEFINED layout

Bits [31:0]

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

4.3.19.3 Accessibility

None.

4.3.20 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, 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, 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].

If RAS System Architecture v1.1 is not implemented, ERRDEVAFF can only describe a group of error records that is affine with a single PE or all the PEs at an affinity level.

Configurations

ERRDEVAFF is present only if the group of error records has affinity with a PE or cluster of PEs. ERRDEVAFF is RES0 otherwise.

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

Attributes

ERRDEVAFF is a 64-bit read-only memory-mapped register located at offset `0xFA8`.

4.3.20.1 Field descriptions

The ERRDEVAFF bit assignments are:

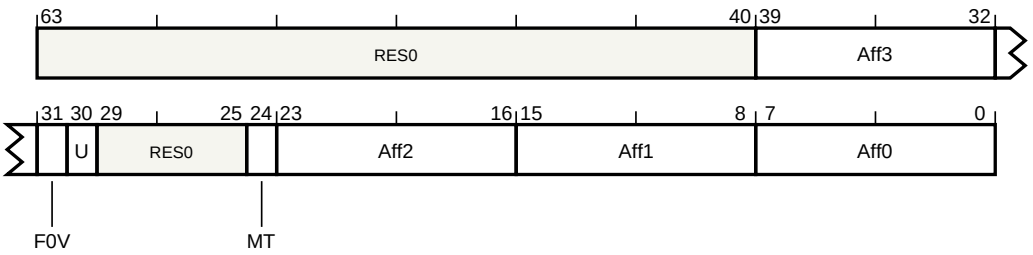

Figure 4.29: ERRDEVAFF

Bits [63:40,29:25]

Reserved. This field is 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.

F0V, bit [31]
Indicates that the ERRDEVAFF.Aff0 field is valid. The defined values of this bit are:

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.

U, bit [30]
Uniprocessor.

When ERRDEVAFF.F0V == 0b1
The MPIDR_EL1.U bit, viewed from the highest Exception level of the associated PE.

Otherwise
Reserved. This bit is UNKNOWN.

MT, bit [24]
Multithreaded.

When ERRDEVAFF.F0V == 0b1
The MPIDR_EL1.MT bit, viewed from the highest Exception level of the associated PE.

Otherwise
Reserved. This bit is UNKNOWN.

Aff2, bits [23:16]
PE affinity level 2.

When affine with a PE or PEs at affinity level 2 or below
The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 2
Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

0bxxxxxxx1	ERRDEVAFF.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	ERRDEVAFF.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	ERRDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	ERRDEVAFF.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	ERRDEVAFF.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	ERRDEVAFF.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	ERRDEVAFF.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.

Otherwise
Indicates whether the PE affinity is at level 3. The defined values of this field are:

0x80	PE affinity is at level 3.

All other values are reserved.

Aff1, bits [15:8]
PE affinity level 1.

When affine with a PE or PEs at affinity level 1 or below
The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 1
Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

`0bxxxxxxx1`	ERRDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
`0bxxxxxx10`	ERRDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
`0bxxxxx100`	ERRDEVAFF.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
`0bxxxx1000`	ERRDEVAFF.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
`0bxxx10000`	ERRDEVAFF.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
`0bxx100000`	ERRDEVAFF.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
`0bx1000000`	ERRDEVAFF.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.

Otherwise
Indicates whether the PE affinity is at level 2. The defined values of this field are:

`0x00`	PE affinity is above level 2 or a subset of level 2.
`0x80`	PE affinity is at level 2.

Aff0, bits [7:0]
PE affinity level 0.

When affine with a PE at affinity level 0
The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

When affine with a sub-set of PEs at affinity level 0
Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs. The defined values of this field are:

`0bxxxxxxx1`	ERRDEVAFF.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
`0bxxxxxx10`	ERRDEVAFF.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
`0bxxxxx100`	ERRDEVAFF.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
`0bxxxx1000`	ERRDEVAFF.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
`0bxxx10000`	ERRDEVAFF.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
`0bxx100000`	ERRDEVAFF.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.

0bx1000000	ERRDEVAFF.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.

Otherwise

Indicates whether the PE affinity is at level 1. The defined values of this field are:

0x00	PE affinity is above level 1 or a subset of level 1.
0x80	PE affinity is at level 1.

4.3.20.2 Accessibility

None.

4.3.21 ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose
Provides discovery information for the component.

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

Attributes
ERRDEVARCH is a 32-bit read-only memory-mapped register located at offset `0xFBC`.

4.3.21.1 Field descriptions

The ERRDEVARCH bit assignments are:

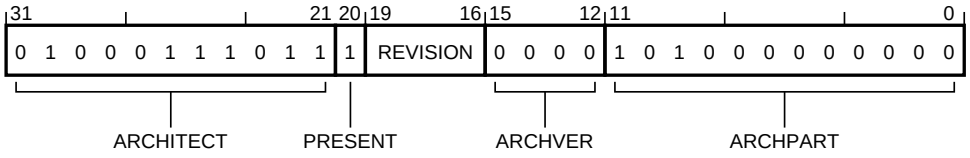

Figure 4.30: ERRDEVARCH

ARCHITECT, bits [31:21]
Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code. The defined values of this field are:

`0x23B`	JEP106 continuation code `0x4`, ID code `0x3B`. Arm Limited.

Other values are defined by the JEDEC JEP106 standard.

This field reads as `0x23B`.

PRESENT, bit [20]
DEVARCH Present. Defines that the DEVARCH register is present. The defined values of this bit are:

`0b0`	Device Architecture information not present.
`0b1`	Device Architecture information present.

This bit reads as `0b1`.

REVISION, bits [19:16]
Revision. Defines the architecture revision of the component. The defined values of this field are:

`0b0000`	RAS System Architecture v1.0.
`0b0001`	RAS System Architecture 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 RAS Common Fault Injection Model Extension.

All other values are reserved.

ARCHVER, bits [15:12]
Architecture Version. Defines the architecture version of the component. The defined values of this field are:

0b0000	RAS System Architecture v1.

All other values are reserved.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0b0000.

ARCHPART, bits [11:0]
Architecture Part. Defines the architecture of the component. The defined values of this field are:

0xA00	RAS System Architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA00.

4.3.21.2 Accessibility

None.

4.3.22 ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

Purpose
Provides discovery information for the component.

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

Attributes
ERRDEVID is a 32-bit read-only memory-mapped register located at offset `0xFC8`.

4.3.22.1 Field descriptions

The ERRDEVID bit assignments are:

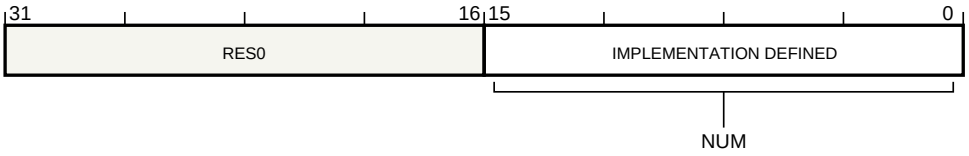

Figure 4.31: ERRDEVID

Bits [31:16]
Reserved. This field is RES0.

NUM, bits [15:0]
Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This 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.

This field reads as an IMPLEMENTATION DEFINED value.

4.3.22.2 Accessibility

None.

4.3.23 ERRERICR0, Error Recovery Interrupt Configuration Register 0

The ERRERICR0 characteristics are:

Purpose
Error Recovery Interrupt configuration register.

Configurations
ERRERICR0 is present only if all of the following are true:

- Any of the following are true:
  - The Error Recovery Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRERICR0 is RES0 otherwise.

ERRERICR0 is architecturally mapped to memory-mapped register ERRIRQCR2[63:0].

ERRERICR0 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRERICR0 is a 64-bit read/write memory-mapped register located at offset `0xE90`.

4.3.23.1 Error Recovery Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Error Recovery Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Error Recovery Interrupt is implemented, recommended layout bit assignments are:

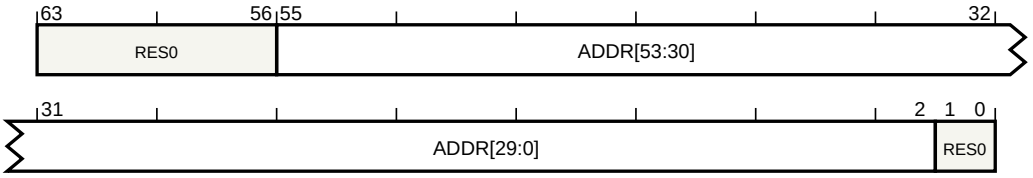

Figure 4.32: ERRERICR0 Error Recovery Interrupt is implemented, recommended layout

Bits [63:56,1:0]
Reserved. This field is 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.

This field resets to an architecturally UNKNOWN value on a reset.

4.3.23.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

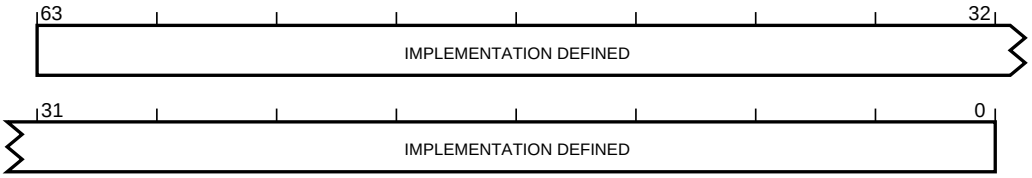

Figure 4.33: ERRERICR0 IMPLEMENTATION DEFINED layout

Bits [63:0]

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

4.3.23.3 Accessibility

None.

4.3.24 ERRERICR1, Error Recovery Interrupt Configuration Register 1

The ERRERICR1 characteristics are:

Purpose
Error Recovery Interrupt configuration register.

Configurations
ERRERICR1 is present only if all of the following are true:

- Any of the following are true:
  - The Error Recovery Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRERICR1 is RES0 otherwise.

ERRERICR1 is architecturally mapped to memory-mapped register ERRIRQCR3[31:0].

ERRERICR1 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRERICR1 is a 32-bit read/write memory-mapped register located at offset `0xE98`.

4.3.24.1 Error Recovery Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Error Recovery Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Error Recovery Interrupt is implemented, recommended layout bit assignments are:

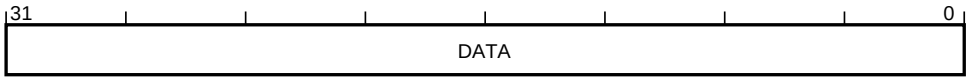

Figure 4.34: ERRERICR1 Error Recovery Interrupt is implemented, recommended layout

DATA, bits [31:0]
Payload for the message signaled interrupt. This field resets to an architecturally UNKNOWN value on a reset.

4.3.24.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

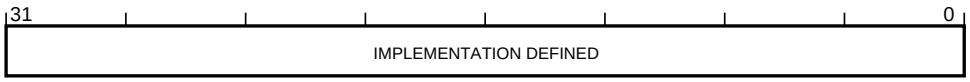

Figure 4.35: ERRERICR1 IMPLEMENTATION DEFINED layout

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

4.3.24.3 Accessibility

None.

4.3.25 ERRERICR2, Error Recovery Interrupt Configuration Register 2

The ERRERICR2 characteristics are:

Purpose
Error Recovery Interrupt control and configuration register.

Configurations
ERRERICR2 is present only if all of the following are true:

- Any of the following are true:
  - The Error Recovery Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRERICR2 is RES0 otherwise.

ERRERICR2 is architecturally mapped to memory-mapped register ERRIRQCR3[63:32].

ERRERICR2 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRERICR2 is a 32-bit read/write memory-mapped register located at offset `0xE9C`.

4.3.25.1 Error Recovery Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Error Recovery Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Error Recovery Interrupt is implemented, recommended layout bit assignments are:

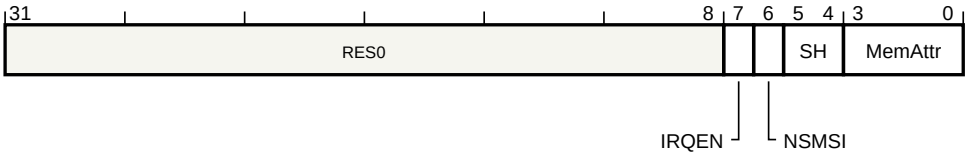

Figure 4.36: ERRERICR2 Error Recovery Interrupt is implemented, recommended layout

Bits [31:8]
Reserved. This field is RES0.

IRQEN, bit [7]
Message signaled interrupt enable.

When the component supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

`0b0`	Disabled.
`0b1`	Enabled.

This bit resets to `0b0` on a reset.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [6]
Security attribute. Defines the physical address space for message signaled interrupts.

When the component supports configuring the Security attribute for message signaled interrupts, and the component does not allow Non-secure writes to ERRERICR2
The possible values of this bit are:

0b0	Secure.
0b1	Non-secure.

This bit resets to an IMPLEMENTATION DEFINED value on a reset.

When the component allows Non-secure writes to ERRERICR2
The Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0.

Otherwise
The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [5:4]
Shareability.

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

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.

This field resets to an architecturally UNKNOWN value on a reset.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [3:0]
Memory type.

When the component supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

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

All other values are reserved.

This field resets to an architecturally UNKNOWN value on a reset.

Note:

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

Otherwise

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

This field is RES0.

4.3.25.2 IMPLEMENTATION DEFINED layout

Configurations

Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

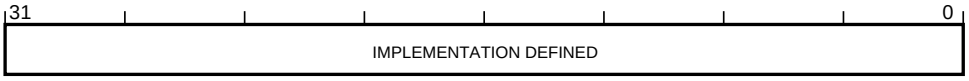

Figure 4.37: ERRERICR2 IMPLEMENTATION DEFINED layout

Bits [31:0]

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

4.3.25.3 Accessibility

None.

4.3.26 ERRFHICR0, Fault Handling Interrupt Configuration Register 0

The ERRFHICR0 characteristics are:

Purpose
Fault Handling Interrupt configuration register.

Configurations
ERRFHICR0 is present only if all of the following are true:

- Any of the following are true:
  - The Fault Handling Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRFHICR0 is RES0 otherwise.

ERRFHICR0 is architecturally mapped to memory-mapped register ERRIRQCR0[63:0].

ERRFHICR0 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRFHICR0 is a 64-bit read/write memory-mapped register located at offset `0xE80`.

4.3.26.1 Fault Handling Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Fault Handling Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Fault Handling Interrupt is implemented, recommended layout bit assignments are:

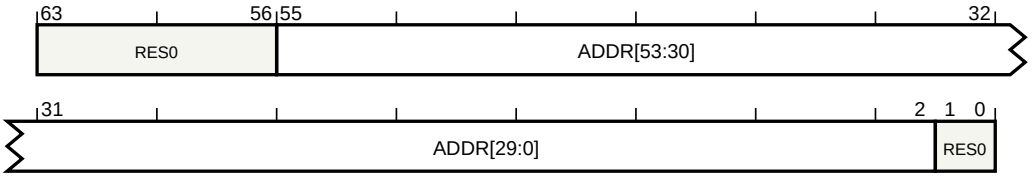

Figure 4.38: ERRFHICR0 Fault Handling Interrupt is implemented, recommended layout

Bits [63:56,1:0]
Reserved. This field is 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.

This field resets to an architecturally UNKNOWN value on a reset.

4.3.26.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

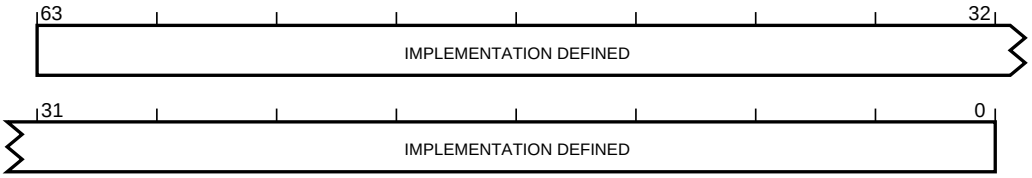

Figure 4.39: ERRFHICR0 IMPLEMENTATION DEFINED layout

Bits [63:0]

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

4.3.26.3 Accessibility

None.

4.3.27 ERRFHICR1, Fault Handling Interrupt Configuration Register 1

The ERRFHICR1 characteristics are:

Purpose
Fault Handling Interrupt configuration register.

Configurations
ERRFHICR1 is present only if all of the following are true:

- Any of the following are true:
  - The Fault Handling Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRFHICR1 is RES0 otherwise.

ERRFHICR1 is architecturally mapped to memory-mapped register ERRIRQCR1[31:0].

ERRFHICR1 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRFHICR1 is a 32-bit read/write memory-mapped register located at offset `0xE88`.

4.3.27.1 Fault Handling Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Fault Handling Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Fault Handling Interrupt is implemented, recommended layout bit assignments are:

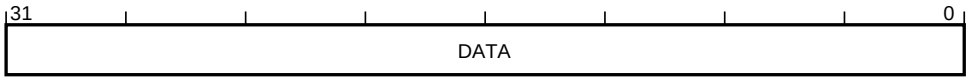

Figure 4.40: ERRFHICR1 Fault Handling Interrupt is implemented, recommended layout

DATA, bits [31:0]
Payload for the message signaled interrupt. This field resets to an architecturally UNKNOWN value on a reset.

4.3.27.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

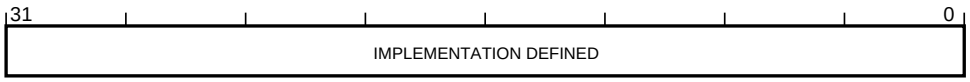

Figure 4.41: ERRFHICR1 IMPLEMENTATION DEFINED layout

Bits [31:0]
This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

4.3.27.3 Accessibility

None.

4.3.28 ERRFHICR2, Fault Handling Interrupt Configuration Register 2

The ERRFHICR2 characteristics are:

Purpose
Fault Handling Interrupt control and configuration register.

Configurations
ERRFHICR2 is present only if all of the following are true:

- Any of the following are true:
  - The Fault Handling Interrupt is implemented.
  - The implementation does not use the recommended layout for the ERRIRQCR<n> registers.
- Interrupt configuration registers are implemented.

ERRFHICR2 is RES0 otherwise.

ERRFHICR2 is architecturally mapped to memory-mapped register ERRIRQCR1[63:32].

ERRFHICR2 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRFHICR2 is a 32-bit read/write memory-mapped register located at offset `0xE8C`.

4.3.28.1 Fault Handling Interrupt is implemented, recommended layout

Configurations
Defined only if all of the following are true:

- The Fault Handling Interrupt is implemented.
- The implementation uses the recommended layout for the ERRIRQCR<n> registers.

The Fault Handling Interrupt is implemented, recommended layout bit assignments are:

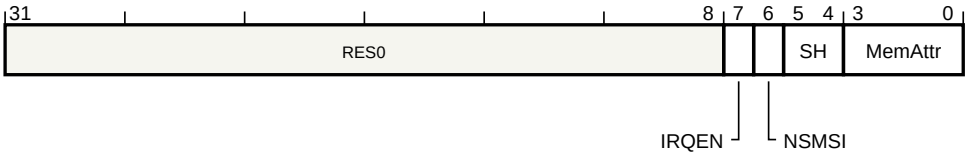

Figure 4.42: ERRFHICR2 Fault Handling Interrupt is implemented, recommended layout

Bits [31:8]
Reserved. This field is RES0.

IRQEN, bit [7]
Message signaled interrupt enable.

When the component supports disabling message signaled interrupts
Enables generation of message signaled interrupts. The possible values of this bit are:

`0b0`	Disabled.
`0b1`	Enabled.

This bit resets to `0b0` on a reset.

Otherwise
Message signaled interrupts are always enabled.

This bit is RES0.

NSMSI, bit [6]
Security attribute. Defines the physical address space for message signaled interrupts.

When the component supports configuring the Security attribute for message signaled interrupts, and the component does not allow Non-secure writes to ERRFHICR2
The possible values of this bit are:

0b0	Secure.
0b1	Non-secure.

This bit resets to an IMPLEMENTATION DEFINED value on a reset.

When the component allows Non-secure writes to ERRFHICR2
The Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0.

Otherwise
The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

This bit is RES0.

SH, bits [5:4]
Shareability.

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

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.

This field resets to an architecturally UNKNOWN value on a reset.

Otherwise
The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

This field is RES0.

MemAttr, bits [3:0]
Memory type.

When the component supports configuring the memory type for message signaled interrupts
Defines the memory type and attributes for message signaled interrupts. The possible values of this field are:

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

All other values are reserved.

This field resets to an architecturally UNKNOWN value on a reset.

Note:

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

Otherwise

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

This field is RES0.

4.3.28.2 IMPLEMENTATION DEFINED layout

Configurations

Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

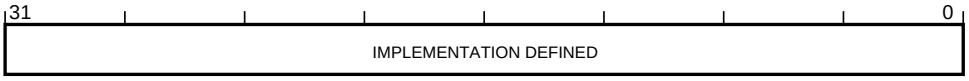

Figure 4.43: ERRFHICR2 IMPLEMENTATION DEFINED layout

Bits [31:0]

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

4.3.28.3 Accessibility

None.

4.3.29 ERRGSR, Error Group Status Register

The ERRGSR characteristics are:

Purpose
Shows the status for the records in the group.

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

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.

Attributes
ERRGSR is a 64-bit read-only memory-mapped register located at offset `0xE00`.

4.3.29.1 Field descriptions

The ERRGSR bit assignments are:

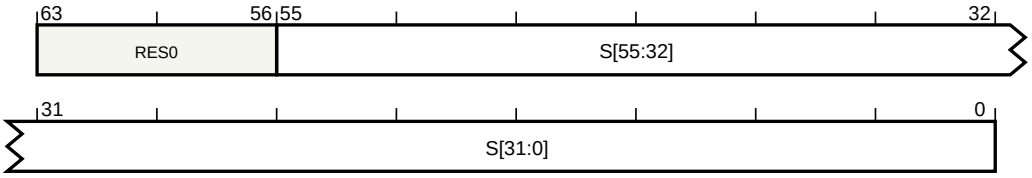

Figure 4.44: ERRGSR

Bits [63:56]
Reserved. This field is RES0.

S[*m*], bit [*m*], for *m* = 0 to 55
The status for error record <*m*>. A read-only copy of ERR<m>STATUS.V.

When error record <*m*> is implemented, and error record <*m*> supports this type of reporting
The defined values of this bit are:

`0`	No error.
`1`	One or more errors.

If the Common Fault Injection Model is implemented, up-to 24 records can be implemented meaning bits [55:24] are RES0.

Otherwise
Reserved. This bit is RES0.

4.3.29.2 Accessibility

None.

4.3.30 ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

Purpose
Defines the implementer of the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRIIDR is present. ERRIIDR is RES0 if not present.

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

Attributes
ERRIIDR is a 32-bit read-only memory-mapped register located at offset `0xE10`.

4.3.30.1 Field descriptions

The ERRIIDR bit assignments are:

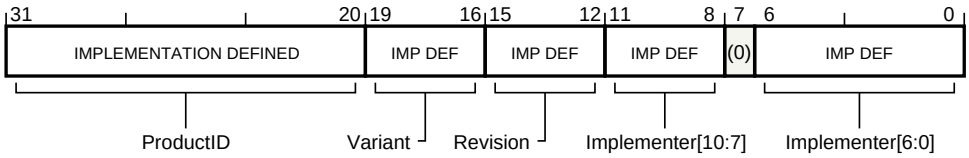

Figure 4.45: ERRIIDR

ProductID, bits [31:20]
Part number, bits [11:0]. The part number is selected by the designer of the component.

Matches the {ERRPIDR1.PART_1,ERRPIDR0.PART_0} fields, if ERRPIDR0 and ERRPIDR1 are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Variant, bits [19:16]
Component major revision.

ERRIIDR.Variant defines either a variant of the component defined by ERRIIDR.ProductID, or the major revision of the component.

When defining a major revision, ERRIIDR.Variant and ERRIIDR.Revision together form the revision number of the component, with ERRIIDR.Variant being the most significant part and ERRIIDR.Revision the least significant part. When a component is changed, ERRIIDR.Variant or ERRIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component. If ERRIIDR.Variant is increased then ERRIIDR.Revision should be set to `0b0000`.

Matches the ERRPIDR2.REVISION field, if ERRPIDR2 is also present.

This field reads as an IMPLEMENTATION DEFINED value.

Revision, bits [15:12]
Component minor revision.

When a component is changed:

- If ERRIIDR.Variant and ERRIIDR.Revision together form the revision number of the component then:
  - ERRIIDR.Variant or ERRIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.
  - If Variant is increased then Revision should be set to `0b0000`.

- Otherwise, ERRIIDR.Revision is increased to ensure that software can differentiate the different revisions of the component.

Matches the ERRPIDR3.REVAND field, if ERRPIDR3 is also present.

This field reads as an IMPLEMENTATION DEFINED value.

Implementer, bits [11:8,6:0]
JEDEC-assigned JEP106 identification code. ERRIIDR[11:8] is the JEP106 bank identifier minus 1 and ERRIIDR[6:0] is the JEP106 identification code for the designer of the component. 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.

ERRIIDR[11:8] matches ERRPIDR4.DES_2 and ERRIIDR[6:0] match the {ERRPIDR2.DES_1,ERRPIDR1.DES_0} fields, if ERRPIDR{1,2,4} are also present.

This field reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 bank is 5, and the JEP106 identification code is `0x3B`, meaning ERRIIDR[11:0] has the value `0x43B`.

Zero is not a valid JEP106 identification code, meaning a value of zero for ERRIIDR indicates this register is not implemented.

Bit [7]
Reserved. This bit is RES0.

4.3.30.2 Accessibility

None.

4.3.31 ERRIMPDEF<*n*>, IMPLEMENTATION DEFINED Register <0-191>

The ERRIMPDEF<0-191> characteristics are:

Purpose

IMPLEMENTATION DEFINED RAS extensions.

Configurations

ERRIMPDEF<*n*> is present if all of the following are true:

- The RAS Common Fault Injection Model Extension is not implemented.
- ERRDEVID.NUM <= 32.

It is IMPLEMENTATION DEFINED whether ERRIMPDEF<*n*> is present.

ERRIMPDEF<*n*> is RES0 if not present.

Attributes

ERRIMPDEF<*n*> is a 64-bit read/write memory-mapped register located at offset `0x800` + 8×*n*.

4.3.31.1 Field descriptions

The ERRIMPDEF<0-191> bit assignments are:

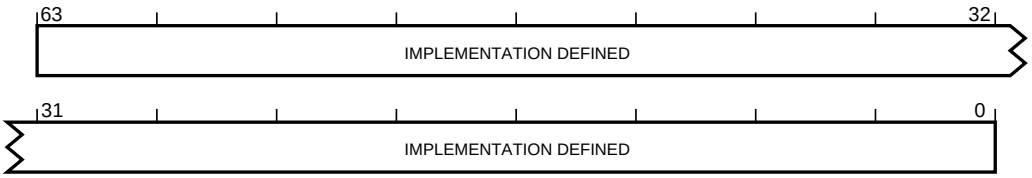

Figure 4.46: ERRIMPDEF<*n*>

Bits [63:0]

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

4.3.31.2 Accessibility

None.

4.3.32 ERRIRQCR<*n*>, Generic Error Interrupt Configuration Register

The ERRIRQCR<0-15> characteristics are:

Purpose

The ERRIRQCR<*n*> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

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.

Configurations

ERRIRQCR<*n*> is present only if the interrupt configuration registers are implemented. ERRIRQCR<*n*> is RES0 otherwise.

ERRIRQCR<*n*> is implemented only as part of a memory-mapped group of error records.

Attributes

ERRIRQCR<*n*> is a 64-bit read/write memory-mapped register located at offset `0xE80` + 8×*n*.

4.3.32.1 Field descriptions

The ERRIRQCR<0-15> bit assignments are:

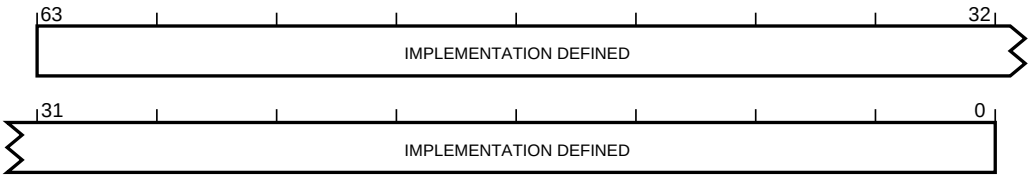

Figure 4.47: ERRIRQCR<*n*>

Bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

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

4.3.32.2 Accessibility

None.

4.3.33 ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose
Interrupt status register.

Configurations
ERRIRQSR is present only if interrupt configuration registers are implemented. ERRIRQSR is RES0 otherwise.

ERRIRQSR is architecturally mapped to memory-mapped register ERRIRQCR15.

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

Attributes
ERRIRQSR is a 64-bit read/write memory-mapped register located at offset `0xEF8`.

4.3.33.1 Recommended layout

Configurations
Defined only if the implementation uses the recommended layout for the ERRIRQCR<n> registers.

The recommended layout bit assignments are:

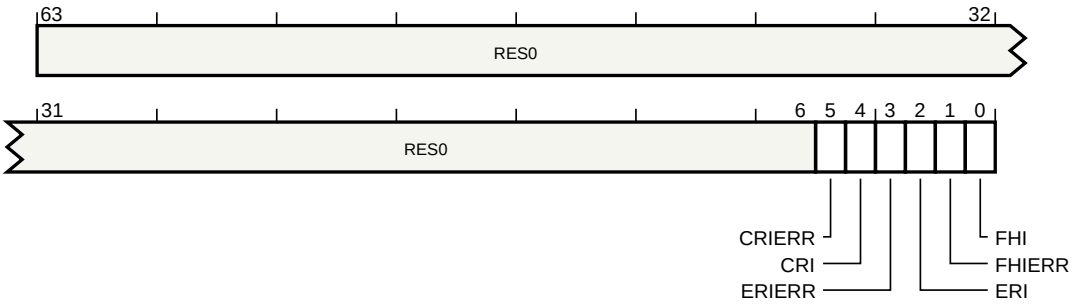

Figure 4.48: ERRIRQSR recommended layout

Bits [63:6]
Reserved. This field is RES0.

CRIERR, bit [5]
Critical Error Interrupt error.

When the Critical Error Interrupt is implemented
The possible values of this bit are:

`0b0`	Critical Error Interrupt write has not returned an error since this bit was last cleared to zero.
`0b1`	Critical Error Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a reset.

Otherwise
Reserved. This bit is RES0.

CRI, bit [4]
Critical Error Interrupt write in progress.

When the Critical Error Interrupt is implemented
The defined values of this bit are:

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.

This bit is read-only.

Note:

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Otherwise
Reserved. This bit is RES0.

ERIERR, bit [3]
Error Recovery Interrupt error.

When the Error Recovery Interrupt is implemented
The possible values of this bit are:

0b0	Error Recovery Interrupt write has not returned an error since this bit was last cleared to zero.
0b1	Error Recovery Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a reset.

Otherwise
Reserved. This bit is RES0.

ERI, bit [2]
Error Recovery Interrupt write in progress.

When the Error Recovery Interrupt is implemented
The defined values of this bit are:

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.

This bit is read-only.

Note:

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Otherwise
Reserved. This bit is RES0.

FHIERR, bit [1]
Fault Handling Interrupt error.

When the Fault Handling Interrupt is implemented
The possible values of this bit are:

0b0	Fault Handling Interrupt write has not returned an error since this bit was last cleared to zero.
0b1	Fault Handling Interrupt write has returned an error since this bit was last cleared to zero.

This bit is read/write-one-to-clear.

This bit resets to an architecturally UNKNOWN value on a reset.

Otherwise
Reserved. This bit is RES0.

FHI, bit [0]
Fault Handling Interrupt write in progress.

When the Fault Handling Interrupt is implemented
The defined values of this bit are:

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.

This bit is read-only.

Note:

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Otherwise
Reserved. This bit is RES0.

4.3.33.2 IMPLEMENTATION DEFINED layout

Configurations
Defined only if the implementation does not use the recommended layout for the ERRIRQCR<n> registers.

The IMPLEMENTATION DEFINED layout bit assignments are:

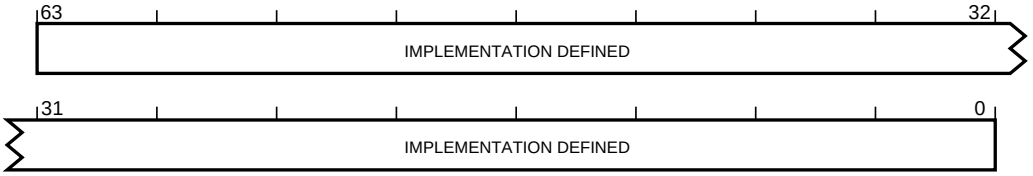

Figure 4.49: ERRIRQSR IMPLEMENTATION DEFINED layout

Bits [63:0]

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

4.3.33.3 Accessibility

None.

4.3.34 ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRPIDR0 is present. ERRPIDR0 is RES0 if not present.

ERRPIDR0 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRPIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFE0`.

4.3.34.1 Field descriptions

The ERRPIDR0 bit assignments are:

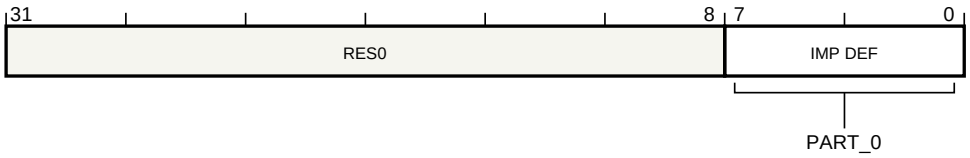

Figure 4.50: ERRPIDR0

Bits [31:8]
Reserved. This field is 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, 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, 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 reads as an IMPLEMENTATION DEFINED value.

4.3.34.2 Accessibility

None.

4.3.35 ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRPIDR1 is present. ERRPIDR1 is RES0 if not present.

ERRPIDR1 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRPIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFE4`.

4.3.35.1 Field descriptions

The ERRPIDR1 bit assignments are:

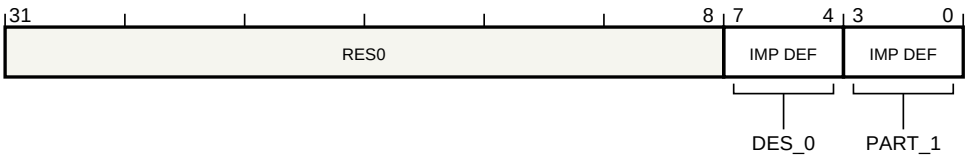

Figure 4.51: ERRPIDR1

Bits [31:8]
Reserved. This field is 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 reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

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, 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, 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 reads as an IMPLEMENTATION DEFINED value.

4.3.35.2 Accessibility

None.

4.3.36 ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRPIDR2 is present. ERRPIDR2 is RES0 if not present.

ERRPIDR2 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRPIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFE8`.

4.3.36.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

Figure 4.52: ERRPIDR2 the component uses a 12-bit part number

Bits [31:8]
Reserved. This field is 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. If ERRPIDR2.REVISION is increased then ERRPIDR3.REVAND should be set to `0b0000`.

This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]
JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

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 reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

4.3.36.2 The component uses a 16-bit part number

Configurations

Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

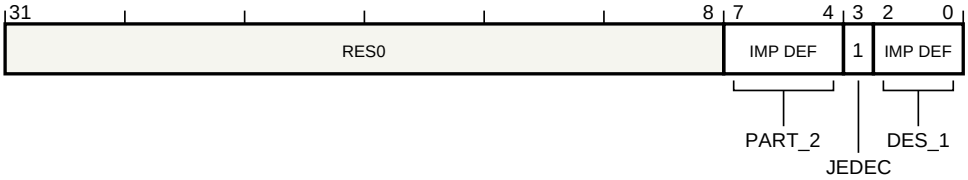

Figure 4.53: ERRPIDR2 the component uses a 16-bit part number

Bits [31:8]

Reserved. This field is 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, 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, 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 reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used. This bit reads as `0b1`.

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 reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

4.3.36.3 Accessibility

None.

4.3.37 ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRPIDR3 is present. ERRPIDR3 is RES0 if not present.

ERRPIDR3 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRPIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFEC`.

4.3.37.1 The component uses a 12-bit part number

Configurations
Defined only if the component uses a 12-bit part number.

The the component uses a 12-bit part number bit assignments are:

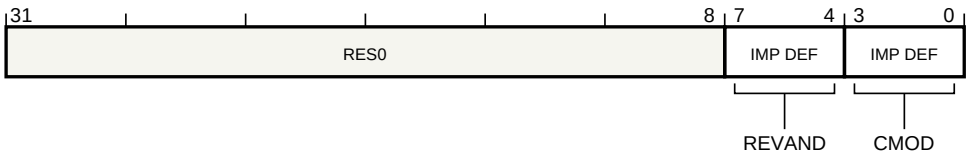

Figure 4.54: ERRPIDR3 the component uses a 12-bit part number

Bits [31:8]
Reserved. This field is 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. If ERRPIDR2.REVISION is increased then ERRPIDR3.REVAND should be set to `0b0000`.

This field reads as an IMPLEMENTATION DEFINED value.

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.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

4.3.37.2 The component uses a 16-bit part number

Configurations
Defined only if the component uses a 16-bit part number.

The the component uses a 16-bit part number bit assignments are:

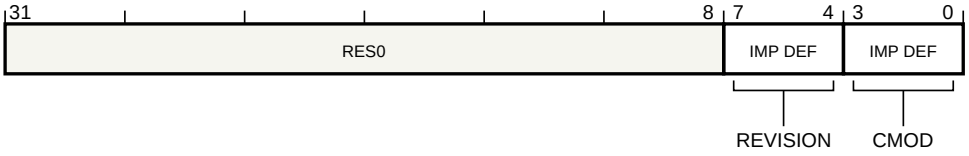

Figure 4.55: ERRPIDR3 the component uses a 16-bit part number

Bits [31:8]
Reserved. This field is 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 reads as an IMPLEMENTATION DEFINED value.

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.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

4.3.37.3 Accessibility

None.

4.3.38 ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose
Provides discovery information about the component.

Configurations
It is IMPLEMENTATION DEFINED whether ERRPIDR4 is present. ERRPIDR4 is RES0 if not present.

ERRPIDR4 is implemented only as part of a memory-mapped group of error records.

Attributes
ERRPIDR4 is a 32-bit read-only memory-mapped register located at offset `0xFD0`.

4.3.38.1 Field descriptions

The ERRPIDR4 bit assignments are:

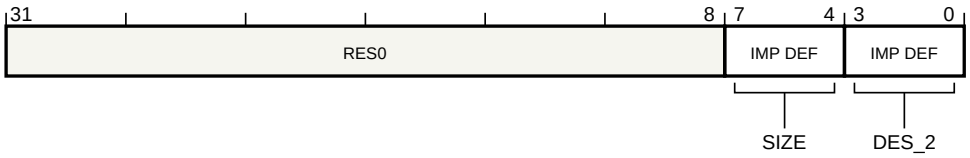

Figure 4.56: ERRPIDR4

Bits [31:8]
Reserved. This field is RES0.

SIZE, bits [7:4]
Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of `0b0000` means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{\text{ERRPIDR4.SIZE}}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field reads as an IMPLEMENTATION DEFINED value.

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 reads as an IMPLEMENTATION DEFINED value.

Note:

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

4.3.38.2 Accessibility

None.

Glossary

Asynchronous exception

Asynchronous exceptions are also known as interrupts. In the Armv8 architecture, an asynchronous exception is one for which any of the following apply:

- The exception is not generated as a result of direct execution or attempted execution of the instruction stream.
- The return address presented to the exception handler is not guaranteed to indicate the instruction that caused the exception.
- The exception is imprecise.

Availability

Readiness for correct service.

Baseboard Management Controller

A PE dedicated to system control and monitoring.

BIST

Built-in self-test

Built-in self-test

A mechanism that permits a machine to test itself.

Catastrophic failure

A failure with harmful consequences that are orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery.

CE

Correctable or Corrected Error

Completer

An agent in a computing system that responds to and completes a memory transaction initiated by a Requester.

Contained or containable error

An error that is not uncontained or uncontainable.

Containment

Limiting or preventing the silent propagation of an error. Arm recommends that the scope to which an error is contained is specified.

Correctable or Corrected Error

An error that is detected by hardware and that hardware can correct / has corrected.

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.

Double fault

A second error that is detected when the PE is in the process of handling a first error condition.

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.

Error synchronization event

One of:

- Executing an `ESB` instruction.
- Taking an exception to an Exception level using AArch64, FEAT_IESB is implemented, and either:
  - The appropriate SCTLR_ELx.IESB bit is `0b1`.
  - FEAT_DoubleFault is implemented, the Exception level is EL3, and SCTLR_EL3.NMEA is `0b1`.

- Executing an Exception Return instruction at an Exception level using AArch64, FEAT_IESB is implemented, and either:
  - The appropriate SCTLR_ELx.IESB bit is `0b1`.
  - FEAT_DoubleFault is implemented, the Exception level is EL3, and SCR_EL3.NMEA is `0b1`.

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

General-purpose registers

The registers that the base instructions use for processing:

- In AArch32 state the general-purpose registers are R0-R14.
- In AArch64 state the general-purpose registers are R0-R30.

Generic Interrupt Controller

Arm system architecture interrupt controller for IRQ and FIQ interrupt exceptions.

GIC

Generic Interrupt Controller

Hardware fault

A fault that originates in, or affects, hardware.

Imprecise exception

An exception that is not precise.

Infected

Being in error.

Interrupt

In a PE context, an asynchronous exception. There are three interrupt exceptions: IRQ, FIQ and SError. IRQ and FIQ are always precise. In a system architecture context, an asynchronous event sent to a PE or GIC for processing as an interrupt exception.

Isolation

Limiting the impact of an error only to components that actually try to use corrupted data.

Latent 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

Precise exception

An exception where the exception handler receives the state of the PE and the state of the memory system consistent with the PE having executed all of the instructions up to, but not including, the point in the instruction stream where the exception was taken. The state of the PE and the state of the memory do not include instructions that occurred after this point.

Predictive Failure Analysis

Mechanisms to analyze errors and predict future failures.

Processing element (PE)

The abstract machine defined in the Armv8 architecture, as documented in an Arm Architecture Reference Manual. A PE implementation compliant with the Armv8 architecture 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 error

A contained error that must be corrected to allow the correct operation of the system or smaller parts of the system to continue.

Reliability

Continuity of correct service.

Requester

An agent in a computing system that initiates memory transactions.

Restartable error

A contained error that does not immediately impact correct operation. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part.

SDC

Silent Data Corruption

SDC FIT rate

The FIT rate for failures because of SDC.

SDEC

Single device error correction EDAC. This can detect and correct multiple clustered errors in a protection granule, such as the types of errors that might be seen if a protection granule is striped across multiple devices and multiple errors come from a single device.

SECDED

Single error correct, double error detect EDAC. This can detect a single or double bit error and correct a single bit error in a protection granule.

SED

Single error detect EDC. This can detect a single bit error in a protection granule.

SError Interrupt

An asynchronous interrupt in the Armv8 architecture.

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.

Synchronous exception

In the Armv8 architecture, an exception for which all of the following apply:

- The exception is generated as a result of direct execution or attempted execution of an instruction.
- The return address presented to the exception handler is guaranteed to indicate the instruction that caused the exception.
- The exception is precise.

Synchronous External Abort

A synchronous exception in the Armv8 architecture.

System Control Processor

A PE dedicated to system control and monitoring.

Transient fault

A fault that is not persistent.

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 error

A contained error that is not recoverable. Continued correct operation is generally not possible. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part. Systems might use high-level recovery techniques to work around an unrecoverable yet contained error in a component so that the system recovers from the error.