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

Armv8, for the Armv8-A architecture profile

Document number: DDI 0587C.b

Arm Reliability, Availability, and Serviceability (RAS) Specification

Release information

The following changes have been made to this document.

Date	Issue	Confidentiality	Change
March 2017	A	Non-Confidential	First issue.
September 2017	B	Non-Confidential	EAC Release.
December 2017	B.a	Non-Confidential	Updated EAC Release.
October 2018	C.a	Non-Confidential	Initial v8.4 EAC Release.
July 2019	C.b	Non-Confidential	Updated v8.4 Release

Proprietary Notice

English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

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

LES-PRE-20349

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

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

This document describes the RAS Extension, and must be read in conjunction with the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*.

References

This document refers to the following documents.

Reference	Document number	Author	Document name
[1]	ARM DDI 0487	Arm	*Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*

Feedback

Arm welcomes feedback on its documentation.

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

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

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

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

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

RAS techniques reduce unplanned outages because:

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

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

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

The RAS Extension introduces the Error synchronization event and Error Synchronization Barrier (ESB) instruction.

The RAS Extension also introduces a set of System and memory-mapped registers that are specific to the RAS architecture. The memory-mapped registers are described in this specification, and the System registers are described in the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*.

Armv8.4 introduces two architectural features to the RAS Extension: ARMv8.4-RAS, and ARMv8.4-DFE. ARMv8.4-RAS and ARMv8.4-DFE are defined in this specification and in the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*.

1.1 Types of errors

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

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

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

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

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

- They are masked and do not affect the outcome of the system:
  - These are benign or false errors.

- They affect the service interface of the system and cause failure:
  - These are silent data corruptions (SDC).
  - The rate of such failures, measured as the number of failures per billion device-hours of operation, is called the SDC *Failure-in-Time* (FIT) rate.

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

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

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

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

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

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

1.2 Techniques for improving reliability, availability, and serviceability

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

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

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

1.2.1 Fault prevention and fault removal

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

Fault prevention techniques are outside the scope of the architecture.

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

1.2.2 Error handling and recovery

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

- Deferring an error from faults by not taking action immediately, but waiting for either:
  - The fault to be masked later (fault removal).
  - The error to be reported to a consumer of the error (this enables error recovery).
- Preventing further propagation of the error, that is *containing* the error.

- Reducing the severity of the error by invoking a *service failure mode*:
  - These are detected uncorrected errors (DUE).
  - The rate of such failures gives the DUE FIT rate.
  - The type of service failure mode depends on what is acceptable to the service.

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

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

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

1.2.3 Fault handling

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

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

Fault handling and error recovery might be independent agents.

1.3 General taxonomy of errors

1.3.1 Error detection

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

1.3.2 Error propagation

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

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

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

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

1.3.3 Infected, poisoned, containable, and uncontainable

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

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

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

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

—— Note ——————————

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

————————————————

1.4 Fault handling

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

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

2 Armv8-A Error Handling and Recovery Architecture

2.1 Taxonomy of errors in the Armv8-A architecture

2.1.1 Architectural error detection

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

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

The component that detects an error is called a *node*. The nodes included as part of a processor, including an Armv8-A PE, are IMPLEMENTATION DEFINED. An Armv8-A PE implementing the RAS Extension must implement the System register interface to the IMPLEMENTATION DEFINED nodes of a PE. For more information, see Nodes.

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

—— Note ——————————

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

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

2.1.3 Architectural error propagation

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

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

That is, the error is propagated by:

- A store of a corrupt value.

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

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

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

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

2.1.4 Architecturally infected, containable, and uncontainable

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

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

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

Architectural error propagation defines error propagation for a PE.

2.1.5 Architecturally consumed errors

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

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

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

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

2.1.6 Other errors

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

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

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

2.2 Generating error exceptions

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

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

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

It is IMPLEMENTATION DEFINED whether an error that is consumed by hardware speculation or prefetching by a PE, but that is not committed to the architecturally visible state of the PE, generates an External abort exception at the PE.

—— Note ——————————

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

————————————————

An External abort must be taken as one of:

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

It is IMPLEMENTATION DEFINED whether an External abort is taken synchronously or asynchronously for non-speculative:

- Instruction fetches, unless ARMv8.4-RAS is implemented.
- Explicit accesses to memory made by instructions.
- Translation table walks.
- Hardware updates of translation tables.

All other External aborts must be taken asynchronously as an SError interrupt.

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

Uncontainable error (UC)

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

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

Unrecoverable error (UEU)

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

— The error has not been silently propagated.

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

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

— An Error synchronization event must contain the error. See Error synchronization event.

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

Recoverable error (UER)

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

— The error has not been silently propagated.

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

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

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

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

Restartable error (UEO)

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

— The error has not been silently propagated.

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

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

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

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

This taxonomy is shown by Figure 1.

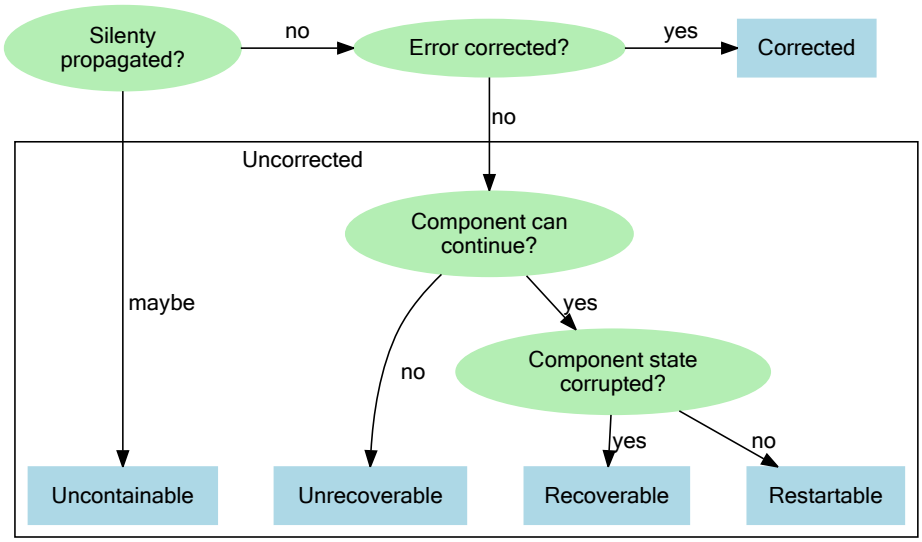


Figure 1: Taxonomy of consumed error types

It is IMPLEMENTATION DEFINED whether the severity of an asynchronous error and the state of the PE are reported when the asynchronous SError interrupt exception is taken. If the severity of an asynchronous error and the state of the PE are not reported, it is further IMPLEMENTATION DEFINED whether the error is reported as uncategorized or with an IMPLEMENTATION DEFINED syndrome. The severity of a synchronous error and the state of the PE are always reported when a synchronous external abort exception is taken. See:

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

It is IMPLEMENTATION DEFINED whether IMPLEMENTATION DEFINED and uncategorized SError interrupts are edge-triggered or level-sensitive. Level-sensitive SError interrupts cannot be synchronized by an Error synchronization event.

The set of error types that can be reported by an implementation is IMPLEMENTATION DEFINED. An error that is not reported as uncategorized or with an IMPLEMENTATION DEFINED syndrome must be reported as:

- Restartable, Recoverable, Unrecoverable, or Uncontainable, if the state of the PE is Restartable.
- Recoverable, Unrecoverable, or Uncontainable, if the state of the PE is Recoverable.
- Unrecoverable or Uncontainable, if the state of the PE is Unrecoverable.
- Uncontainable, if the error is Uncontainable.

Note

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

2.3 Error record System register view

Note

AArch64 Error record System registers are all registers with an ER*_EL1 mnemonic. AArch32 Error record System registers are all registers with an ER* mnemonic. Both of these are defined in the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.*

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

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

To access an error record, software must:

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

Note

See Synchronization and error record accesses.

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

If ARMv8.4-RAS is implemented, any error records accessible through System registers must implement RAS System Architecture v1.1.

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

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

For a given implementation:

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

Table 1: Permitted relaxations for bits in error syndrome registers

Bit that is permitted to be RES0 or RES1	**If always set to zero or always set to one on taking an SError interrupt in all of**		
	ESR_ELx[*x*], $x \in [24:0]$	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

2.4 Error synchronization event

The RAS Extension adds the Error synchronization event operation and the ESB instruction.

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

All Unrecoverable errors must be synchronized by Error synchronization events.

—— Note ——————————

Synchronous abort exceptions are not synchronized by an Error synchronization event.

Error recovery interrupts are asynchronous and are not synchronized by an Error synchronization event.

————————————————

2.4.1 Error synchronization Barrier instruction

The ESB instruction generates an Error synchronization event and additionally might record and then clear a pending maked asynchronous SError interrupt. For details and the encoding of ESB, see the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.*

In AArch32 state:

- ESB might update the DISR or VDISR registers.
- The DISR register can only be accessed at EL1 or above. If EL2 is implemented and HCR.AMO is set to 1, then reads and writes of DISR at Non-secure EL1 access VDISR.
- If the ESB instruction for a physical SError sets DISR.A to 1, then the pending state of the SError is cleared.
- If the ESB instruction for a virtual SError sets VDISR.A to 1, then the pending state of the vSError is cleared.

In AArch64 state:

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

- If the ESB instruction for a physical SError sets DISR_EL1.A to 1, then the pending state of the SError is cleared.
- If the ESB instruction for a virtual SError sets VDISR_EL2.A to 1, then the pending state of the vSError is cleared.

2.4.2 Error synchorinization events and Unrecoverable errors

For Unrecoverable errors, an Error synchronization event guarantees that:

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

—— Note ——————————

- That is, for an Error synchronization event generated by an ESB instruction, the preferred return address of the exception is the ESB instruction address.
- The prioritization of this exception with respect to any other asynchronous exceptions that are pending and unmasked when the Error synchronization event completes is IMPLEMENTATION DEFINED.

————————————————

An Error synchronization event generated by an ESB instruction further guarantees that, if such a physical SError is pended by the Error synchronization event generated by the ESB instruction, or was pending before the Error synchronization event, and the physical SError interrupt is masked at the current Exception level, all of the following is true:

- The pending SError interrupt is cleared.
- The SError interrupt syndrome is recorded in DISR_EL1 or DISR.
- The DISR_EL1.A or DISR.A field is set to 1 to indicate the SError interrupt was generated in program order before the ESB.

The Error synchronization event contains the error. See Architecturally infected, containable, and uncontainable.

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

- Before the ESB, are either taken before or at the ESB, or recorded in the DISR_EL1 or DISR register by the ESB.
- After the ESB, are taken after the ESB and not recorded in DISR_EL1 or DISR by the ESB.

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

If the Unrecoverable SError interrupt exception is:

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

2.4.3 Error synchronization events and other containable errors

For other types of containable error:

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

An unconsumed SError interrupt taken by 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.

These SError interrupts are always precise.

It is IMPLEMENTATION DEFINED whether IMPLEMENTATION DEFINED and uncategorized SError interrupts are containable or Uncontainable.

2.4.4 Other physical SError interrupts

An Uncontainable error might be taken at an Error synchronization event, or recorded in the DISR_EL1 or DISR register by an ESB instruction, but this is not architecturally required. If an Uncontainable error is not taken and not recorded in the DISR_EL1 or DISR register, then it remains pending.

It is IMPLEMENTATION DEFINED whether IMPLEMENTATION DEFINED and uncategorized SError interrupts are containable or Uncontainable.

2.4.5 ESB and Virtual SError interrupts

If EL2 is implemented and enabled in the current Security state, then an ESB instruction executed at EL0 or EL1 also synchronizes a pending virtual SError interrupt when any of:

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

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

- If the virtual SError interrupt is unmasked at the current Exception level, it is taken before the completion of the ESB instruction.
- If the virtual SError interrupt is masked at the current Exception level:
  - HCR_EL2.VSE or HCR.VA is cleared to 0.
  - The virtual SError interrupt syndrome from VSESR_EL2 or VDFSR is recorded in VDISR_EL2 or VDISR, and VDISR_EL2.A or VDISR.A is set to 1 to indicate the SError interrupt was pending prior to the execution of the ESB instruction.

—— Note ——————————

This happens in parallel with the Error synchronization event for physical SError interrupts.

————————————————

2.4.6 Extension for barrier at exception entry and exit

The Armv8.2-IESB architectural feature adds a control bit to each SCTLR_ELx register to insert an *implicit* Error Synchronization event at exception entry and exception return. For the register field descriptions, see the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.*

For each Exception level, ELx, other than EL0, the implicit Error synchronization event at each exception or DCPSx instruction that is taken to ELx guarantees that Unrecoverable errors that are generated before the

exception have pended an SError interrupt exception. If an Unrecoverable SError interrupt that can be synchronized is pended by, or was pending before the exception, then:

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

  —— Note ——————————

  That is, either the SError interrupt is taken before the original exception, or the preferred return address for the SError interrupt is the original exception vector address.

  ————————————————

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

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

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

  —— Note ——————————

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

  ————————————————

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

The implicit form of Error synchronization event:

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

—— Note ——————————

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

- Behave as if the SError interrupt was taken before the implicit Error synchronization event, if the SError interrupt was not masked.
  - For the exception entry case, taking the SError interrupt in place of the exception. ESR_EL*y*.IESB is zero and the value reported in ESR_ELy.AET must correctly indicate whether execution is restartable from the preferred return address for the SError interrupt in ELR_ELy.
  - For the exception return case, the ESR_EL*y*.IESB bit might be zero, but, otherwise the behavior is the same.
- Take another exception before the SError interrupt. In this case, the SError interrupt remains pending.

Arm recommends the SError interrupt is prioritized over other exceptions.

When ARMv8.4-DFE is implemented, and the Effective value of SCR_EL3.NMEA is 1:

- SCTLR_EL3.IESB is ignored and its Effective value is 1.
- For exceptions taken to EL3, Arm recommends that the implicit Error synchronization event is inserted before taking the exception.

2.5 Taking external abort and SError interrupt exceptions to AArch64

2.5.1 Target Exception level

The *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile* states that for SError interrupt and synchronous external abort exceptions taken to AArch64 state, the default target Exception level is:

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

However, the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile* also defines:

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

2.5.2 AArch64 syndrome registers on taking an error exception

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

2.5.3 Multiple SError interrupts

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

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

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

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

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

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

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

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

2.6 Taking external abort and SError interrupt exceptions to AArch32

2.6.1 Target mode

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

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

However:

- If EL3 is implemented and using AArch32 and SCR.EA is set to 1:
  - All SError interrupt and synchronous external 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 or HCR.TGE is set to 1, all SError interrupts from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.
  - If HCR.TEA or HCR.TGE is set to 1, all synchronous external abort exceptions from EL0 and EL1 are taken to Hyp mode, using vector offset `0x14`.

2.6.2 AArch32 syndrome registers on taking an error exception

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

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

2.6.3 Multiple SError interrupts

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

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

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

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

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

2.7 Virtual SError interrupts

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

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

3 RAS System Architecture

3.1 Recording errors

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

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

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

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

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

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

3.1.1 Detected uncorrected errors in data

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

3.1.2 Corrected errors in data

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

Arm recommends that hardware also records enough information to:

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

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

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

3.1.3 Other sources of error and warnings

Other sources of error and warning are possible in a system. Within the Armv8-A architecture these are signaled to a PE using an error recovery or fault handling interrupt.

3.1.4 Error types

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

Corrected error

The error was detected and corrected. The error has not been silently propagated. In normal circumstances, the error no longer infects the state of the node. In rare IMPLEMENTATION DEFINED circumstances, the error might remain latent in the node. The node continues to operate.

Deferred error

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

Uncorrected error

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

— *Unrecoverable*: The error has not been silently propagated.

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

The following two sections list additional sub-types.

This taxonomy is shown by Figure 2.

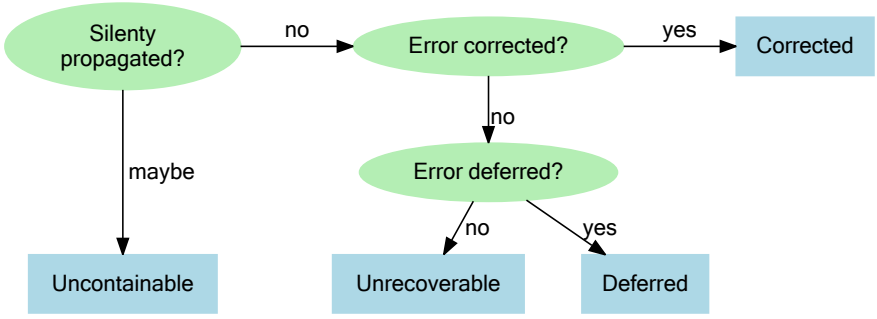


Figure 2: Taxonomy of producer error types

Additional subtypes for an Uncorrected error produced at the node

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

Latent

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

Signaled

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

—— **Note** ——————————

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

——————————————————

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

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

Additional subtypes for an Uncorrected error consumed at the node

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

Restartable

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

Recoverable

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

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

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

3.1.5 Software faults

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

3.2 Standard error record

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

- 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.
  - Optional Corrected Error counters for software to poll the rate of Corrected errors.

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

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

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

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

—— Note ——————————

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

————————————————

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

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

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

The format of the error record registers is the same for both access mechanisms.

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

3.2.1 Nodes

A component might implement one or more nodes. The architecture defines the following common features for a node:

Error detection and correction

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

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

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting and logging are disabled.

Fault handling interrupt

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

See also Fault handling interrupt.

Corrected error counting

It is IMPLEMENTATION DEFINED whether a node that implements error correction implements a counter for counting Corrected errors. Software might poll the error counter or initialize the counter with a 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 and Uncorrected errors are counted by the Corrected error counter.

See also Standard format Corrected error counter.

RAS Timestamp Extension

Optional part of the RAS System Architecture v1.1.

See also RAS Timestamp Extension.

In-band Uncorrected error signaling (external aborts)

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

See also In-band error signaling (external aborts).

Uncorrected error recovery and handling interrupt

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

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

See also Error recovery interrupt.

Critical Error handling

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

See also Critical error interrupt.

Records

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

See also Writing the error record.

A node might implement some or all of these features. The 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.

A node has a single ERR<n>FR and a single ERR<n>CTLR register. If the node implements multiple records, each record has the same features and all records share the controls.

—— Note ——————————

If a component requires multiple sets of controls, the component must implement multiple nodes.

————————————————

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

3.2.2 Multiple records per node

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

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

The multiple error records used by a single node are sequentially indexed starting from the first record for the node. For each record other than the first record for the node, the ERR<n>FR.ED field is RAZ, bits [63:2] of the ERR<n>FR register are RES0, and the ERR<n>CTLR register is RES0.

Error records are accessed through groups of error records, corresponding to one or more nodes. That is, a group consists of the error records of one or more nodes and each node implements one or more error records. This group might be sparsely populated, and locations relating to unimplemented error records are RAZ/WI, meaning that they have an ERR<n>FR register that reads as zero. See Nodes.

An example of a group containing four error records owned by three nodes might be arranged as shown in Figure 3:

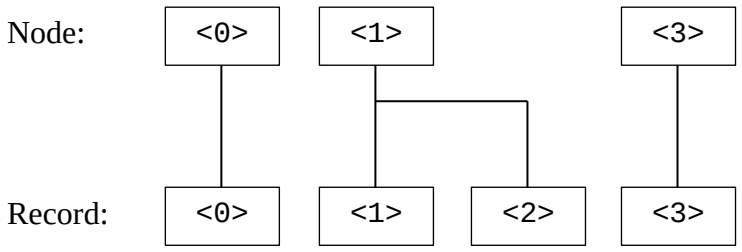


Figure 3: A group containing four error records owned by three nodes

- Node <0> owns a single error record: <0>. ERR0FR describes the features for this node, and ERR0CTRL contains the controls for this node.
- Node <1> owns two error records: <1> and <2>.
  - ERR1FR describes the features for this node, and ERR1CTRL contains the controls for this node.
  - ERR2CTRL is not implemented and ERR2FR.ED is RAZ.
- Node <3> owns a single error record: <3>. ERR3FR describes the features for this node, and ERR3CTRL contains the controls for this node.

3.2.3 Writing the error record

When a new error is detected, the node:

- Modifies ERR<n>STATUS.{CE, DE, UE, UET} to indicate the type of the new detected error. See Error types and priorities.
- Does one of the following:
  - Overwrites the error record with the syndrome for the new error.
  - Keeps the syndrome for the previous error.

- Counts the error, if it is a Corrected Error and a counter is implemented.

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

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

Error types and priorities

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

Table 2: Encoding the highest priority error

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

Prioritizing errors, RAS System Architecture v1.0

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

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

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

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

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

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

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

0 Count CE if a counter is implemented. Keep the previous error syndrome. If the counter overflows, or if no counter is implemented, ERR<n>STATUS.OF is set to 1.

1 Count CE. If ERR<n>STATUS.OF == 1 before the CE is counted, keep the previous syndrome. Otherwise record the new error syndrome.

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

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

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

Table 3: RAS System Architecture v1.0 rules for overwriting error records

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

Prioritizing errors, RAS System Architecture v1.1

When recording an error in the error record, ERR<n>STATUS.OF is set to 1 if any of:

- A corrected error counter is implemented, an error is counted, and the counter overflows.
- A corrected error counter is not implemented, a corrected error is recorded, and ERR<n>STATUS.V was previously set to 1.
- A type of error other than a corrected error is recorded and ERR<n>STATUS.V was previously set to 1.

Otherwise, ERR<n>STATUS.OF is unchanged.

As in RAS System Architecture v1.0, it is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted by the Corrected error counter.

In Table 4, the row and column headings use the mnemonics from Table 2 and the following additional abbreviations are used:

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

WO Overwrite and overflow. Record the new error syndrome. ERR<n>STATUS.OF is set to 1.

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

If no corrected error counter is implemented, **CW** behaves the same as **W**, and **CWO** and **CO** behave the same as **O**.

Otherwise, a corrected error counter is implemented, and:

CW Count and overwrite. Record the new error syndrome and count the error. If counting the error causes unsigned overflow of the counter, set ERR<n>STATUS.OF to 1.

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 1.

Table 4: RAS System Architecture v1.1 rules for overwriting error records

Previous error type	New detected error type					
	CE	**DE**	**UEO**	**UER**	**UEU**	**UC**
-	CW	W	W	W	W	W
CE	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

Overwriting the error syndrome

When an old record is overwritten:

- Bits shown as X in Table 2 for the new error are unchanged.
- The ERR<n>STATUS.{ER, PN, IERR, SERR} syndrome fields are written with the syndrome for the new error.
- If there is an address syndrome for the new error, ERR<n>STATUS.AV is set to 1 and the address is written to ERR<n>ADDR. Otherwise ERR<n>STATUS.AV is set to 0.
- If the RAS Timestamp Extension is implemented, a timestamp is recorded in ERR<n>MISC3 and ERR<n>STATUS.MV is set to 1.
- 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 1.
- 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 0 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 MV bit is unchanged.
  - Otherwise the MV bit is cleared to zero.
- ERR<n>STATUS.V is set to 1.

Keeping the previous error syndrome

When an old record is kept:

- ERR<n>STATUS.{PN, IERR, SERR},ERR<n>ADDR, and ERR<n>STATUS.AV are unchanged. If the new error is an Uncorrected error, then ERR<n>STATUS.UET is unchanged.
- 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 1.

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

Detecting multiple errors

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

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

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

The error types implemented at a node

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

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

3.2.4 Security and Virtualization

Access to the Error System register view of error record registers can be controlled using Trap exceptions. See the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*.

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

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

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

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

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

- The error records are visible to both Non-secure and Secure accesses.

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

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

3.2.5 Synchronization and error record accesses

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

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

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

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

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

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

3.3 Error recovery interrupt

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

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

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

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

3.4 Fault handling interrupt

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

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

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

A *Corrected error event* is either:

- When a counter overflows and sets a counter overflow flag to 1, if the node implements a Corrected error counter. See Corrected error counting. It is UNPREDICTABLE whether the following are Corrected error events:
  - A software write that sets the counter overflow flag.
  - When a counter overflows and the overflow flag was previously set to 1.
- Each detected Corrected error, otherwise.

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

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

3.5 In-band error signaling (external aborts)

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

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

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

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

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

3.6 Critical error interrupt

Support for critical error conditions and critical error interrupts at a node is IMPLEMENTATION DEFINED.

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

An example of a critical error is one where the node has entered a 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 must be signalled to a secondary error controller that has its own private resources in order to log the error.

For a given node, the critical error interrupt is implemented if ERR<n>FR.CI != 0b00.

For a given node, if the critical error interrupt is implemented, then the error recovery interrupt must also be implemented.

The critical error interrupt is enabled when ERR<n>CTLR.CI is set to 1.

If the critical error interrupt is implemented, when a critical error condition is recorded the node sets ERR<n>STATUS.CI to 1, regardless of whether the critical error interrupt is enabled or disabled.

If the critical error interrupt is implemented and disabled, when a critical error condition is detected, the node records the critical error as an Uncontained error.

Classifying the critical error condition as an Uncontained error if the critical error interrupt is not enabled has the effect of causing the node to generate an error recovery interrupt. The node also sets ERR<n>STATUS.CI to 1. If the critical error interrupt is enabled, it is IMPLEMENTATION DEFINED how the error is classified at the node. The critical error flag is set to 1 in addition to the other syndrome information for the error, which is handled in the normal way.

3.7 Standard format Corrected error counter

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

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and a counter overflow flag is set.

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

Note

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

See Corrected error counting.

3.8 Error recovery, fault handling, and critical error signaling

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

For an Arm *Generic Interrupt Controller* (GIC) version 3 or above, 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.

It is IMPLEMENTATION DEFINED whether each node uses independent interrupts for error recovery, fault handling, and critical errors, or whether it shares any of these interrupts with other nodes.

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

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

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

If the interrupt is edge-triggered, it is generated by the node when it is enabled in ERR<n>CTLR and the appropriate event occurs:

- The fault handling interrupt is generated when any of the following occurs:
  - Fault handling interrupts on Deferred and Uncorrected errors are enabled, and an error is detected that was not corrected.
  - Fault handling interrupts on Corrected errors are enabled and a Corrected error event occurs. For a definition of Corrected error events, see Fault handling interrupt.
- The error recovery interrupt is generated when any of the following occurs:
  - Error recovery interrupts on Uncorrected errors are enabled, and an error is detected that is neither corrected nor deferred.
  - Error recovery interrupts on Deferred errors are enabled, and an error is detected and deferred.
- The critical error interrupt is generated while critical error interrupts are enabled, and the node records an error, setting ERR<n>STATUS.CI to 1. The critical error interrupt is asserted even if the ERR<n>STATUS.CI was already set to 1.

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

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.

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

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

3.9 Error Recovery reset

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

This specification defines two classes of reset:

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

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

For a PE, the Error Recovery reset might be implemented by the Warm reset defined by the Armv8-A architecture. If Warm reset is implemented, it must preserve the error records in the PE.

3.10 RAS Timestamp Extension

The RAS Timestamp Extension is an optional part of RAS System Architecture v1.1.

The RAS Timestamp Extension provides a standard mechanism for timestamping error records.

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

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

The timestamp uses either the system Generic Timer counter or an IMPLEMENTATION DEFINED timebase.

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

Other than when IMPLEMENTATION DEFINED conditions apply, the timebase must:

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

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

The Common Fault Injection Model Extension is an optional part of RAS System Architecture v1.1.

Other fault injection mechanisms are permitted.

The Common Fault Injection Model Extension cannot be accessed using AArch32 System registers.

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.

The Common Fault Injection Model Extension fakes the detection of an error at a node.

A faked error detection results in the node signaling the appropriate ones of the fault handling interrupt, error recovery interrupt and in-line error response, according to the type of faked error.

The data is not corrupted by the Common Fault Injection Model Extension.

The Common Fault Injection Model Extension supports generating a subset of the error types supported by the node.

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

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

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

If a node is not capable of detecting an error type, then it does not support injecting that error type. However, it should be noted that most of the ERR<n>FR fields define whether the controls for the node's behavior on detecting an error are implemented, not whether the node is capable of detecting a given error type.

For a given node, the Common Fault Injection Model Extension is disabled if ERR<n>CTLR.ED is writable and set to 0.

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.

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

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 an UNPREDICTABLE 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 node enters a fault injection state.

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

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.

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.

When the Error Generation Counter decrements to/past zero, the node enters a fault injection state.

When the node is in fault injection state, on the next access to the node, the node:

- Fakes detection of the error type(s) described by ERR<n>PFGCTL.
- Records and, if required, reports the injected error.
- Leaves fault injection state.

When an injected error is recorded, the node writes the ERR<n>STATUS.{V, UE, CE, DE, UET} fields according to the error type described by ERR<n>PFGCTL. If ERR<n>PFGCTL defines multiple error types, it is UNPREDICTABLE which is recorded.

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.

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.

If a single node has multiple records, then only the first record has fault injection registers.

If a single node has multiple error records and any of ERR<n>PFGF.{SYN, AV, MV} for the first 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 must be recorded in the first record. Otherwise, the injected error might be recorded in any of the multiple records.

Note

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

If the node implements fault handling, error recovery and critical error interrupts 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.

If the node implements fault handling, error recovery and critical error interrupts 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.

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 an UNPREDICTABLE 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 node reenters a fault injection state.

4 RAS Extension Registers

4.1 Memory-mapped view

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

A memory-mapped component might implement several error records in a group, relating to one or more nodes. 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. A memory-mapped group also implements a group status register.

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

Table 5 describes an alternative 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.

Table 5: Standard memory-mapped view of a single error record

Offset	Size	Register	See
0x000	64	ERR<n>FR	Error Feature Register
0x008	64	ERR<n>CTLR	Error Control Register
0x010	64	ERR<n>STATUS	Error Record Primary Syndrome Register
0x018	64	ERR<n>ADDR	Error Record Address Register
0x020	64	ERR<n>MISC0	Error Record Miscellaneous Registers
0x028	64	ERR<n>MISC1	
0x030	64	ERR<n>MISC2	
0x038	64	ERR<n>MISC3	

The error records in a memory-mapped component might be accessible only through that component, or might be shared and accessible through any of:

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

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:

- Reads and writes of unallocated locations are reserved accesses.
- 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.

- Reads of WO locations are reserved accesses.
- Writes to RO locations are reserved accesses.

The architecture requires reserved accesses to be implemented as 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.

The memory access sizes that are supported by the memory-mapped component are as described for other memory-mapped components in the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*, except that it is IMPLEMENTATION DEFINED whether a word-aligned 32-bit access to either half of a doubleword-aligned 64-bit register is supported if there is no PE in the system that supports AArch32.

4.2 Reset values

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

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

4.3 Writes to ERR<*n*>STATUS

After reading an ERR<n>STATUS register, software must clear the valid bits in the register to allow new errors to be recorded. During this period a new error might have overwritten the syndrome for the previously read error. If the register were read/write, then this would be lost. To prevent this, a write, or part of a write is ignored if fields appear to have been updated. For more information see ERR<n>STATUS.

Some of the fields in ERR<n>STATUS are also defined as UNKNOWN where certain combinations of the ERR<n>STATUS.{V, DE, UE} status fields are zero. The rules for writes to ERR<n>STATUS allow a node to implement such a filed as a fixed read-only value.

For example, when RAS System Architecture v1.1 is implemented, a write to ERR<n>STATUS when V is set to 1 must either result in the V field being cleared to zero, or be ignored. Since all fields in ERR<n>STATUS, other than ERR<n>STATUS.{AV, V, MV}, usually read as UNKNOWN values when the V field is zero, this means those fields can be implemented as read-only if applicable.

For compatibility between implementations, software must attempt to clear those fields when invalidating the ERR<n>STATUS register. 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, as described in Common Fault Injection Model Extension.

4.3.1 Pseudocode

```
// ERRSTATUS[] (assignment form)
// =============================
// For a system register, n = UInt(ERRSELR_EL1.SEL)
ERRSTATUS[integer n] = bits(32) w

    STATUS = _ERRSTATUS[n];                                      // previous value (physical register)
    c = (STATUS[n]<31:19> AND NOT(w<31:19>)):Zeros(3):w<15:0>; // candidate value

    if HaveRASSysArchv1p1() then                                 // RAS System Architecture v1.1
        if !IsZero(c.<V,UE,OF,CE,DE>) then                       // - ignore write if V/UE/DE/CE/OF
           c = STATUS;                                           //   set
    else                                                         // RAS System Architecture v1.0
        if c.OF == '1' then c.<UE,DE,CE> = STATUS.<UE,DE,CE>;  // - do not clear UE/DE/CE if OF set
        if !IsZero(c.<UE,DE,CE>) then c.V = STATUS.V;            // - do not clear V if UE/DE/CE set
        if (c.UE != '0' ||                                       // - do not clear syndrome if not
            (STATUS.UE == '0' && c.DE != '0') ||                 //   clearing highest priority error
            (STATUS.<UE,DE> == '00' && c.CE != '00')) then
            c.<AV,ER,MV,PN,CI,UET,IERR,SERR> = STATUS.<AV,ER,MV,PN,CI,UET,IERR,SERR>;

    _ERRSTATUS[n] = c;

    return;
```

4.4 Error Record registers, including memory mapped view

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.

4.4.1 Register index

Using AArch32 System registers

Table 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

Using AArch64 System registers

Table 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

Use	To Access	Access	Description
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

RAS, error record group

Table 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
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	ERR<irq>CR0, Error Interrupt Configuration Register 0
0xE80+8×*n*	R/W	64	ERRIRQCR<n>	Generic Error Interrupt Configuration Register
0xE88	R/W	32	ERRFHICR1	ERR<irq>CR1, Error Interrupt Configuration Register 1
0xE8C	R/W	32	ERRFHICR2	ERR<irq>CR2, Error Interrupt Configuration Register 2
0xE90	R/W	64	ERRERICR0	ERR<irq>CR0, Error Interrupt Configuration Register 0
0xE98	R/W	32	ERRERICR1	ERR<irq>CR1, Error Interrupt Configuration Register 1
0xE9C	R/W	32	ERRERICR2	ERR<irq>CR2, Error Interrupt Configuration Register 2
0xEA0	R/W	64	ERRCRICR0	ERR<irq>CR0, Error Interrupt Configuration Register 0
0xEA8	R/W	32	ERRCRICR1	ERR<irq>CR1, Error Interrupt Configuration Register 1
0xEAC	R/W	32	ERRCRICR2	ERR<irq>CR2, Error Interrupt Configuration Register 2
0xEF8	R/W	64	ERRIRQSR	Error Interrupt Status Register
0xFA8	RO	64	ERRDEVAFF	Device Affinity Register
0xFBC	RO	32	ERRDEVARCH	Device Architecture Register
0xFC8	RO	32	ERRDEVID	Device Configuration Register
0xFD0	RO	32	ERRPIDR4	Peripheral Identification Register 4

Offset	Access	Size	Register	Description
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

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

The ERR<*irq*>CR0 characteristics are:

Purpose

Interrupt configuration register.

In the name of this register, <*irq*> is the abbreviation for the interrupt, one of:

- FHI for the fault-handling interrupt.
- ERI for the error-recovery interrupt.
- CRI for the critical error interrupt.

Configurations

Present only if interrupt configuration registers use the recommended format. RES0 otherwise.

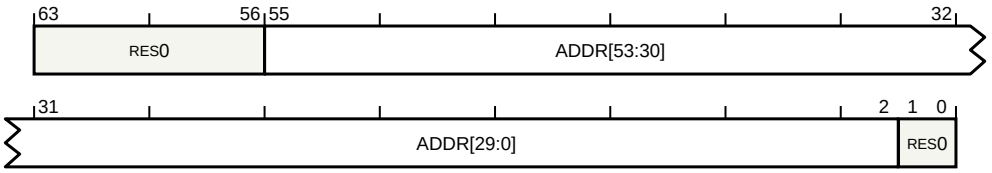
Attributes

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

- Offset `0xEA0` for ERRCRICR0.
- Offset `0xE90` for ERRERICR0.
- Offset `0xE80` for ERRFHICR0.

Field descriptions

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



Bits [63:56,1:0]

Reserved. This field is RES0.

ADDR, bits [55:2]

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

The size of a physical address is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

This field resets to an architecturally UNKNOWN value on a reset.

Accessibility

None.

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

The ERR<*irq*>CR1 characteristics are:

Purpose

Interrupt configuration register.

In the name of this register, <*irq*> is the abbreviation for the interrupt, one of:

- FHI for the fault-handling interrupt.
- ERI for the error-recovery interrupt.
- CRI for the critical error interrupt.

Configurations

Present only if interrupt configuration registers use the recommended format. RES0 otherwise.

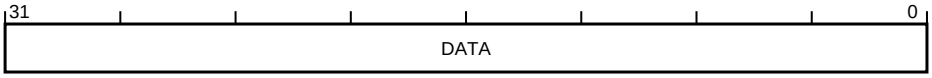
Attributes

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

- Offset `0xEA8` for ERRCRICR1.
- Offset `0xE98` for ERRERICR1.
- Offset `0xE88` for ERRFHICR1.

Field descriptions

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



DATA, bits [31:0]

Payload for a message signaled interrupt. This field resets to an architecturally UNKNOWN value on a reset.

Accessibility

None.

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

The ERR<*irq*>CR2 characteristics are:

Purpose

Interrupt configuration register.

In the name of this register, <*irq*> is the abbreviation for the interrupt, one of:

- FHI for the fault-handling interrupt.
- ERI for the error-recovery interrupt.
- CRI for the critical error interrupt.

Configurations

Present only if interrupt configuration registers use the recommended format. RES0 otherwise.

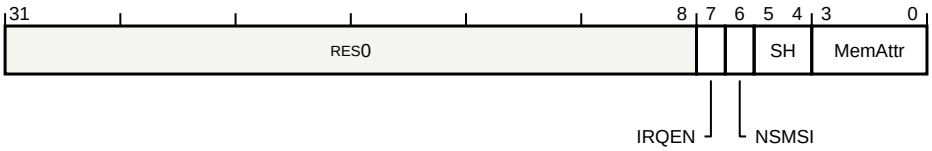
Attributes

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

- Offset `0xEAC` for ERRCRICR2.
- Offset `0xE9C` for ERRERICR2.
- Offset `0xE8C` for ERRFHICR2.

Field descriptions

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



Bits [31:8]

Reserved. This field is RES0.

IRQEN, bit [7]

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

`0` Disabled.

`1` Enabled.

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

This bit resets to zero on a reset.

NSMSI, bit [6]

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

`0` Secure.

`1` Non-secure.

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

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

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

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

This bit resets to an IMPLEMENTATION DEFINED value.

SH, bits [5:4]

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

`0b00` Not shared.

`0b10` Outer Shareable.

`0b11` Inner Shareable.

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

This field resets to an architecturally UNKNOWN value on a reset.

MemAttr, bits [3:0]

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

`0b0000` Device-nGnRnE.

`0b0001` Device-nGnRE.

`0b0010` Device-nGRE.

`0b0011` Device-GRE.

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

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

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

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

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

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

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

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

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

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

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. See the *Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*.

————————————————

Accessibility

None.

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

The ERR<*n*>ADDR characteristics are:

Purpose

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

Configurations

Present only if error record *n* is implemented. 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

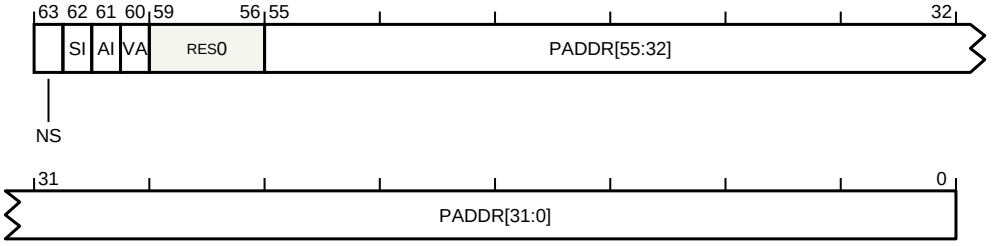
When accessed using System registers, ERR<*n*>ADDR is a 64-bit read/write register accessed using:

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

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

Field descriptions

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



NS, bit [63]

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

`0` The address is Secure.

`1` The address is Non-secure.

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 the NS bit is valid. The possible values of this bit are:

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

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

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

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

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

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

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 the PADDR field is a virtual address. The possible values of this bit are:

- `0` The PADDR field is not a virtual address.
- `1` The PADDR field is a virtual address.

No context information is provided for the virtual address. When this bit is set to 1, ERR<*n*>ADDR.{NS,SI,AI} must read as {0,1,1}.

Support for this bit is optional. If this bit is not implemented and the PADDR field is a virtual address, then ERR<*n*>ADDR.{NS,SI,AI} must 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.

Accessibility

ERR<*n*>ADDR ignores writes if ERR<n>STATUS.AV is set to 1.

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

The ERR<*n*>CTLR characteristics are:

Purpose

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

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

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

Configurations

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.

RES0 otherwise.

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

Attributes

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

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

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

Preface

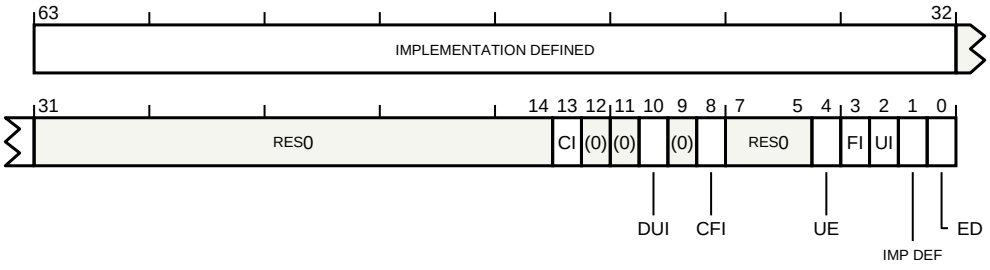
For some controls, it is IMPLEMENTATION DEFINED whether ERR<*n*>CTLR contains a single combined read/write control or separate read and write controls. This register description shows two possible combinations:

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

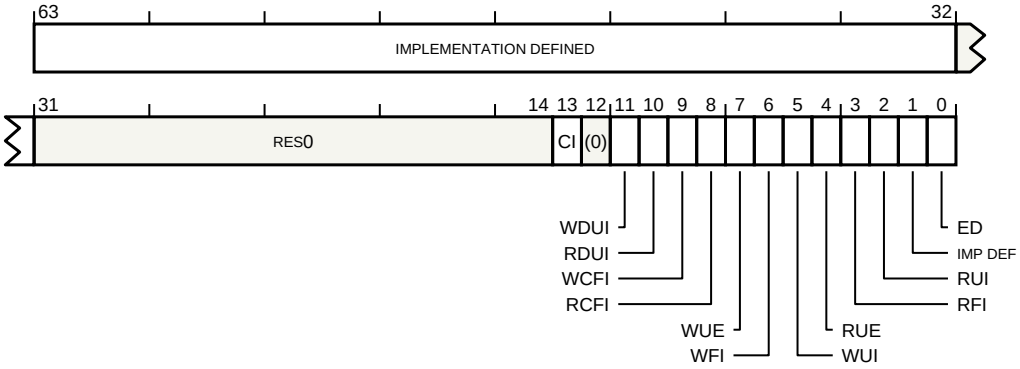
However, this is IMPLEMENTATION DEFINED for each control.

Field descriptions

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



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



Bits [63:32]

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

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

Bits [31:14,12]

Reserved. This field is RES0.

CI, bit [13]

Critical error interrupt enable. 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 is RES0 if the node does not support this control.

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

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

Reserved. This field is RES0.

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

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

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

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

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

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

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

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

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

——Note:——————————

Applies to both reads and writes.

——————————————————

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

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

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

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

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

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

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

Fault handling interrupt for Corrected errors enable.

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 1. See ERR<n>MISC0.
- Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

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

`1` Fault handling interrupt generated for Corrected errors.

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

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

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

——Note:——————————

Applies to both reads and writes.

——————————————————

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

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

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

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

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

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

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

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

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

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

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

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

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

In-band Uncorrected error reporting enable. When enabled, responses to transactions that detect an Uncorrected error that cannot be deferred are signaled 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.

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

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

——**Note:**——————————

Applies to both reads and writes.

————————————————

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

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

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

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

Fault handling interrupt enable.

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 1.
  - Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

0 Fault handling interrupt disabled.

1 Fault handling interrupt enabled.

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

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

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

——**Note:**——————————

Applies to both reads and writes.

————————————————

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

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

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

UI, bit [2], when combined read/write control

Uncorrected error recovery interrupt enable. When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred. The possible values of this bit are:

0 Error recovery interrupt disabled.

1 Error recovery interrupt enabled.

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

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

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

——**Note:**——————————

Applies to both reads and writes.

————————————————

RUI, bit [2], when separate read/write controls

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

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

Bit [1]

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

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

ED, bit [0]

Error reporting and logging enable. When disabled, the node behaves as if error detection and correction are disabled, and no errors are recorded or signaled by the node. Arm recommends that, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection. The possible values of this bit are:

0 Error reporting disabled.

1 Error reporting enabled.

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

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

This bit resets to an IMPLEMENTATION DEFINED value on a Cold reset. 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 0 on Cold reset, meaning errors are not reported from Cold reset. This allows boot software to initialize a node without signaling errors. Software can enable error reporting after the node is initialized. Otherwise, the Cold reset value is IMPLEMENTATION DEFINED. If the Cold reset value is 1, the reset values of other controls in this register are also IMPLEMENTATION DEFINED and should not be UNKNOWN.

————————————————

Accessibility

None.

4.4.7 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 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

Present only if error record *n* is implemented. RES0 otherwise.

Attributes

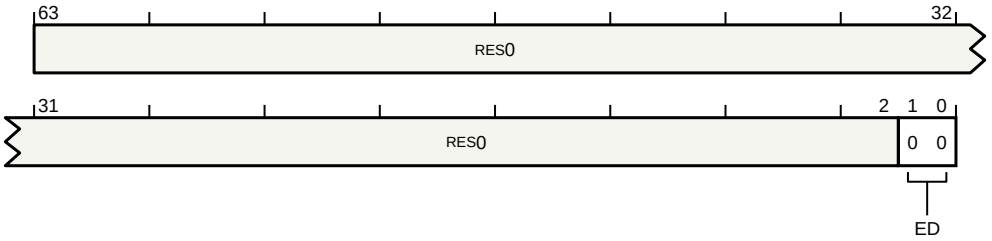
When accessed using System registers, ERR<*n*>FR is a 64-bit read-only register accessed using:

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

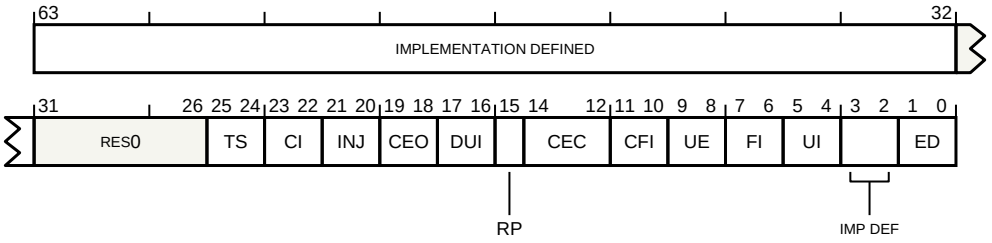
When accessed as memory-mapped registers, ERR<*n*>FR are 64-bit read-only registers located at offset `0x000` + 64×*n*.

Field descriptions

When *n* is not the first record owned by the node, the ERR<*n*>FR bit assignments are:



When *n* is the first record owned by the node, the ERR<*n*>FR bit assignments are:



Bits [63:2], when *n* is not the first record owned by the node

This field is RES0.

Bits [63:32], when *n* is the first record owned by the node

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

Bits [31:26], when *n* is the first record owned by the node

Reserved. This field is RES0.

TS, bits [25:24], when *n* is the first record owned by the node

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 with an affinity to a PE, this means the same timer as is visible through CNTPCT_EL1 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], when *n* is the first record owned by the node

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 feature.

`0b01` Feature always enabled.

`0b10` Feature is controllable.

All other values are reserved.

INJ, bits [21:20], when *n* is the first record owned by the node

Fault Injection Extension. Indicates whether the standard fault injection mechanism is implemented. The defined values of this field are:

`0b00` The node does not support a standard fault injection mechansim.

`0b01` The node implements a standard fault injection mechanism. ERR<n>PFGF provides more information.

All other values are reserved.

CEO, bits [19:18], when *n* is the first record owned by the node

Corrected Error overwrite. 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 1.

`0b01` Counts Corrected errors. If ERR<*m*>STATUS.OF == 1 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 1.

All other values are reserved.

This field reads-as-zeros if no Corrected error counter is implemented. See also Writing the error record.

DUI, bits [17:16], when *n* is the first record owned by the node

Error recovery interrupt for deferred errors. Indicates whether the node implements a control for enabling error recovery interrupts on deferred errors. The defined values of this field are:

`0b00` Does not support feature. ERR<n>CTLR.DUI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.DUI.

`0b11` Feature is controllable using ERR<n>CTLR.WDUI for writes and ERR<n>CTLR.RDUI for reads.

All other values are reserved.

This field is RES0 if ERR<*n*>FR.UI == `0b00`.

RP, bit [15], when *n* is the first record owned by the node

Repeat counter. Indicates whether the node implements a repeat Corrected error counter in ERR<*m*>MISC0 for each error record *m* owned by the node that implements a 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.

This bit is RES0 if ERR<*n*>FR.CEC == `0b00`.

CEC, bits [14:12], when *n* is the first record owned by the node

Corrected Error Counter. Indicates whether the node implements 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], when *n* is the first record owned by the node

Fault handling interrupt for corrected errors. Indicates whether the node implements a control for enabling fault handling interrupts on corrected errors. The defined values of this field are:

`0b00` Does not support feature, ERR<n>CTLR.CFI is RES0.

`0b10` Feature is controllable using ERR<n>CTLR.CFI.

`0b11` Feature is controllable using ERR<n>CTLR.WCFI for writes and ERR<n>CTLR.RCFI for reads.

All other values are reserved.

This field is RES0 if ERR<*n*>FR.FI == `0b00`.

UE, bits [9:8], when *n* is the first record owned by the node

In-band uncorrected error reporting. Indicates whether the node implements in-band uncorrected error reporting (External Aborts), and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

- `0b00` Does not support feature, ERR<n>CTLR.UE is RES0.
- `0b01` Feature always enabled, ERR<n>CTLR.UE is RES0.
- `0b10` Feature is controllable using ERR<n>CTLR.UE.
- `0b11` Feature is controllable using ERR<n>CTLR.WUE for writes and ERR<n>CTLR.RUE for reads.

FI, bits [7:6], when *n* is the first record owned by the node

Fault handling interrupt. Indicates whether the node implements a fault handling interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

- `0b00` Does not support feature, ERR<n>CTLR.FI is RES0.
- `0b01` Feature always enabled, ERR<n>CTLR.FI is RES0.
- `0b10` Feature is controllable using ERR<n>CTLR.FI.
- `0b11` Feature is controllable using ERR<n>CTLR.WFI for writes and ERR<n>CTLR.RFI for reads.

UI, bits [5:4], when *n* is the first record owned by the node

Error recovery interrupt for uncorrected errors. Indicates whether the node implements an error recovery interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

- `0b00` Does not support feature, ERR<n>CTLR.UI is RES0.
- `0b01` Feature always enabled, ERR<n>CTLR.UI is RES0.
- `0b10` Feature is controllable using ERR<n>CTLR.UI.
- `0b11` Feature is controllable using ERR<n>CTLR.WUI for writes and ERR<n>CTLR.RUI for reads.

Bits [3:2], when *n* is the first record owned by the node

This field reads as an IMPLEMENTATION DEFINED value.

ED, bits [1:0]

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

- `0b00` Error record *n* is not the first record owned by the node.
- `0b01` Feature always enabled, ERR<n>CTLR.ED is RES0.
- `0b10` Feature is controllable using ERR<n>CTLR.ED.

All other values are reserved.

Accessibility

None.

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

The ERR<*n*>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

For IMPLEMENTATION DEFINED fields in ERR<*n*>MISC0, writing zero must always be supported to return the error record to an initial 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 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

Present only if error record *n* is implemented. 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 System registers, ERR<*n*>MISC0 is a 64-bit read/write register accessed using:

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

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

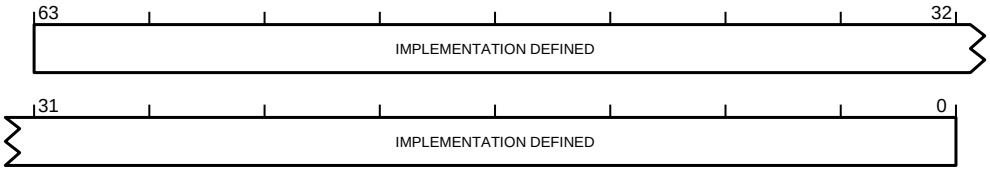
——Note:——————————

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<*q*>CTRL, where *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*.

————————————————

ERR<*n*>MISC0 (contents are IMPLEMENTATION DEFINED)

The ERR<*n*>MISC0 (contents are IMPLEMENTATION DEFINED) bit assignments are:

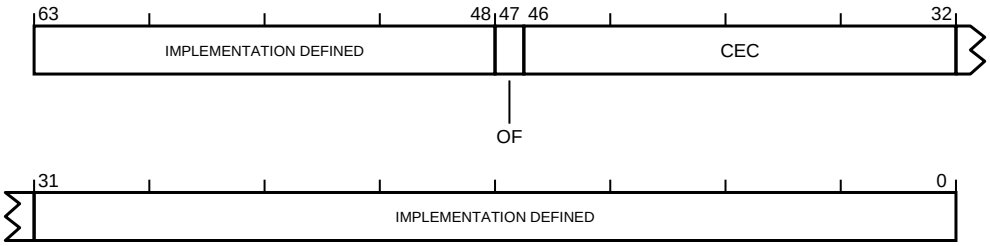


Bits [63:0], when contents are IMPLEMENTATION DEFINED

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

ERR<*n*>MISC0 (standard 16-bit CE counter)

The ERR<*n*>MISC0 (standard 16-bit CE counter) bit assignments are:



Bits [63:48,31:0], when standard 16-bit CE counter

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

OF, bit [47], when standard 16-bit CE counter

Sticky overflow bit. Set to 1 when the Corrected error count field is incremented and wraps through zero. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

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.

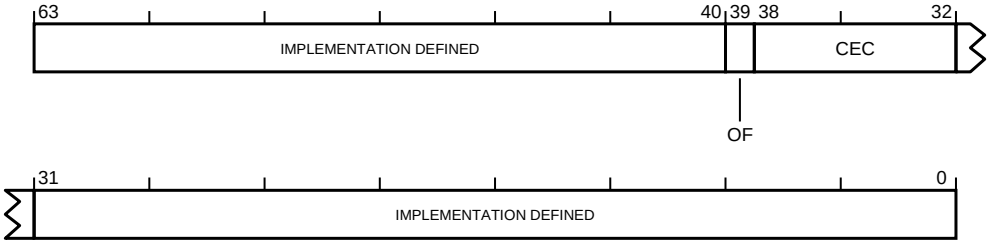
CEC, bits [46:32], when standard 16-bit CE counter

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

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

ERR<*n*>MISC0 (standard 8-bit CE counter)

The ERR<*n*>MISC0 (standard 8-bit CE counter) bit assignments are:



Bits [63:40,31:0], when standard 8-bit CE counter

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

OF, bit [39], when standard 8-bit CE counter

Sticky overflow bit. Set to 1 when the Corrected error count field is incremented and wraps through zero. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

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.

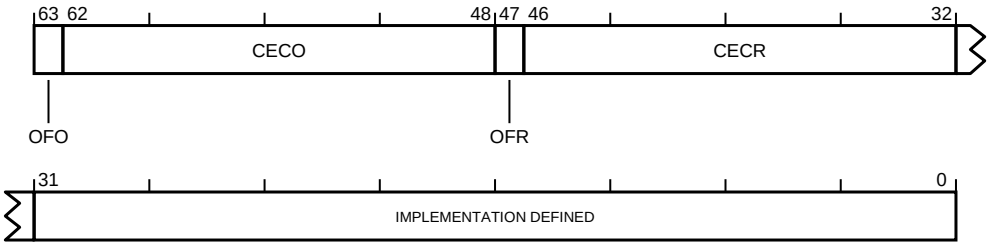
CEC, bits [38:32], when standard 8-bit CE counter

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

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

ERR<*n*>MISC0 (standard 16-bit CE counter pair)

The ERR<*n*>MISC0 (standard 16-bit CE counter pair) bit assignments are:



OFO, bit [63], when standard 16-bit CE counter pair

Sticky overflow bit, other. Set to 1 when the Corrected error count, other, field is incremented and wraps through zero. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

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], when standard 16-bit CE counter pair

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing CECR.

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

OFR, bit [47], when standard 16-bit CE counter pair

Sticky overflow bit, repeat. Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

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], when standard 16-bit CE counter pair

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

This field resets to an 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. CECR is then incremented for each subsequent Corrected Error in the same set and way.

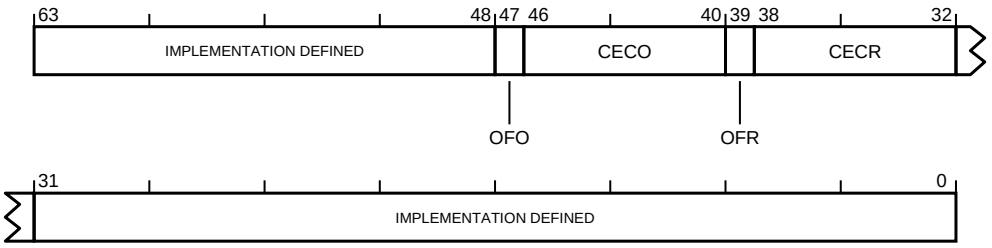
————————————————————

Bits [31:0], when standard 16-bit CE counter pair

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

ERR<*n*>MISC0 (standard 8-bit CE counter pair)

The ERR<*n*>MISC0 (standard 8-bit CE counter pair) bit assignments are:



Bits [63:48,31:0], when standard 8-bit CE counter pair

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

OFO, bit [47], when standard 8-bit CE counter pair

Sticky overflow bit, other. Set to 1 when the Corrected error count, other, field is incremented and wraps through zero. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

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], when standard 8-bit CE counter pair

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing CECR.

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

OFR, bit [39], when standard 8-bit CE counter pair

Sticky overflow bit, repeat. Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

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], when standard 8-bit CE counter pair

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

This field resets to an 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. CECR is then incremented for each subsequent Corrected Error in the same set and way.

————————————————

Accessibility

Reads from ERR<*n*>MISC0 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior. Arm recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes when ERR<n>STATUS.MV is set to 1. This prevents information being lost if an error is detected whilst the previous error is being logged.

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

The ERR<*n*>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

For IMPLEMENTATION DEFINED fields in ERR<*n*>MISC1, writing zero must always be supported to return the error record to an initial 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.

Configurations

Present only if error record *n* is implemented. 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 System registers, ERR<*n*>MISC1 is a 64-bit read/write register accessed using:

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

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

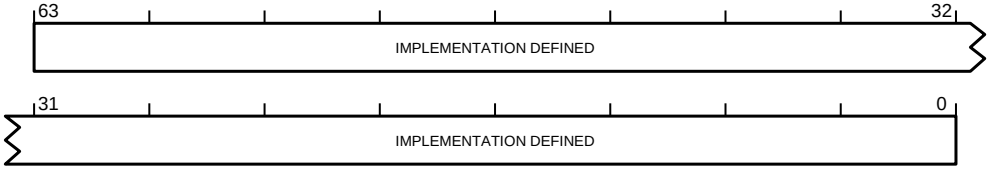
——Note:————————————

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<*q*>CTRL, where *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*.

————————————————————

Field descriptions

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



Bits [63:0]

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

Accessibility

Reads from ERR<*n*>MISC1 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior. Arm recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes when ERR<n>STATUS.MV is set to 1. This prevents information being lost if an error is detected whilst the previous error is being logged.

4.4.10 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 identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

For IMPLEMENTATION DEFINED fields in ERR<*n*>MISC2, writing zero must always be supported to return the error record to an initial 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.

Configurations

Present only if all of the following are true:

- RAS System Architecture v1.1 is implemented.
- Error record *n* is implemented.

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 System registers, 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 memory-mapped registers, ERR<*n*>MISC2 are 64-bit read/write registers located at offset `0x030` + 64×*n*.

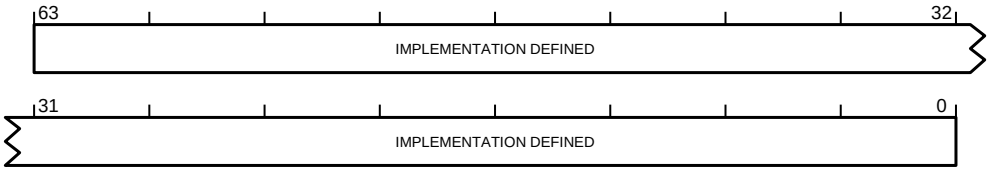
——Note:——————————

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<*q*>CTRL, where *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*.

——————————————————

Field descriptions

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



Bits [63:0]

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

Accessibility

Reads from ERR<*n*>MISC2 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior. Arm recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes when ERR<n>STATUS.MV is set to 1. This prevents information being lost if an error is detected whilst the previous error is being logged.

4.4.11 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 identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

For IMPLEMENTATION DEFINED fields in ERR<*n*>MISC3, writing zero must always be supported to return the error record to an initial 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 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

Present only if all of the following are true:

- RAS System Architecture v1.1 is implemented.
- Error record *n* is implemented.

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 System registers, 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 memory-mapped registers, ERR<*n*>MISC3 are 64-bit read/write registers located at offset `0x038` + 64×*n*.

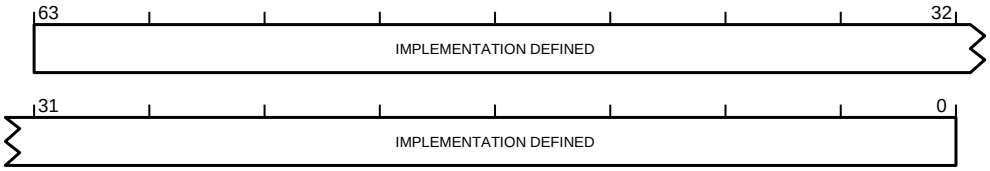
——Note:——————————

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<*q*>CTRL, where *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*.

————————————————

ERR<*n*>MISC3 (contents are IMPLEMENTATION DEFINED)

The ERR<*n*>MISC3 (contents are IMPLEMENTATION DEFINED) bit assignments are:

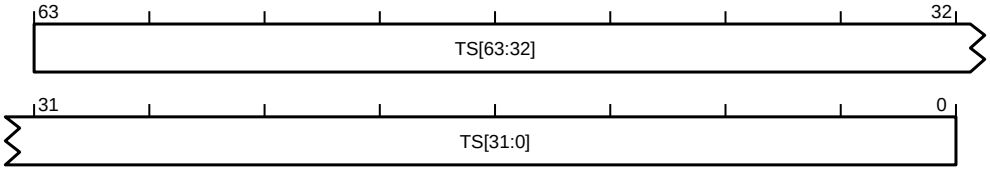


Bits [63:0]

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

ERR<*n*>MISC3 (RAS Timestamp Extension is implemented by the node)

The ERR<*n*>MISC3 (RAS Timestamp Extension is implemented by the node) bit assignments are:



TS, bits [63:0]

Timestamp. Timestamp value recorded when the error was detected. Valid only if ERR<n>STATUS.V == 1.

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<n>FR.TS.

Accessibility

Reads from ERR<*n*>MISC3 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior. Arm recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes when ERR<n>STATUS.MV is set to 1. This prevents information being lost if an error is detected whilst the previous error is being logged.

4.4.12 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

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.

RES0 otherwise.

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

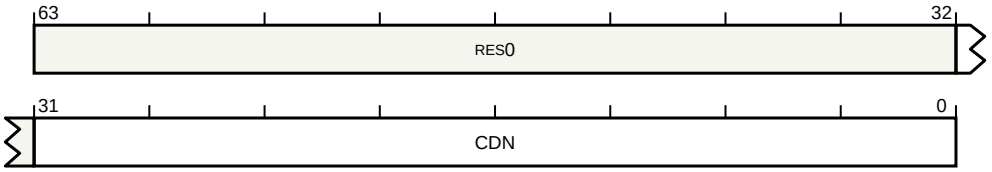
Attributes

When accessed using System registers, 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 memory-mapped registers, ERR<*n*>PFGCDN are 64-bit read/write registers located at offset `0x810` + 64×*n*.

Field descriptions

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



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 == 1.

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

————————————————

Accessibility

None.

4.4.13 ERR<*n*>PFGCTL, Pseudo-fault Generation Control register

The ERR<*n*>PFGCTL characteristics are:

Purpose

Enables controlled fault generation.

Configurations

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.

RES0 otherwise.

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

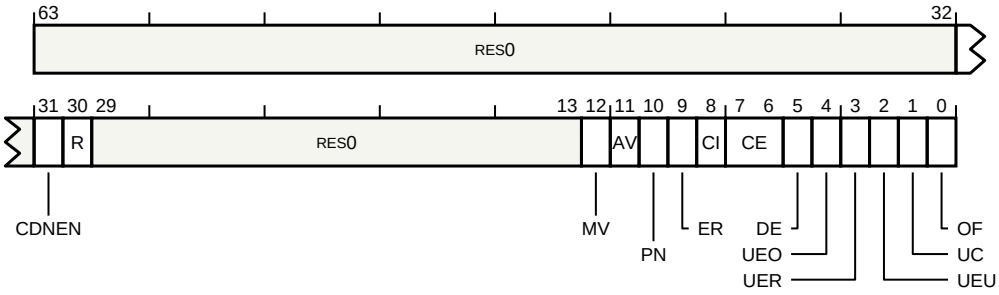
Attributes

When accessed using System registers, 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 memory-mapped registers, ERR<*n*>PFGCTL are 64-bit read/write registers located at offset 0x808 + 64×*n*.

Field descriptions

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



Bits [63:32,29:13]

Reserved. This field is RES0.

CDNEN, bit [31]

Countdown Enable. Controls transfers from the value 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 1 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. 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 Upon reaching 0, the Error Generation Counter will stop.

1 Upon reaching 0, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

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

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

MV, bit [12]

Miscellaneous syndrome. The value 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 0 when an injected error is recorded.

1 ERR<n>STATUS.MV is set to 1 when an injected error is recorded.

This bit reads-as-one 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. This bit is RES0 if the node does not support this control.

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

AV, bit [11]

Address syndrome. The value 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 0 when an injected error is recorded.

1 ERR<n>STATUS.AV is set to 1 when an injected error is recorded.

This bit reads-as-one if the node always sets ERR<n>STATUS.AV to 1 when an injected error is recorded. This bit is RES0 if the node does not support this control.

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

PN, bit [10]

Poison flag. The value 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 0 when an injected error is recorded.

1 ERR<n>STATUS.PN is set to 1 when an injected error is recorded.

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

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

ER, bit [9]

Error Reported flag. The value 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 0 when an injected error is recorded.

1 ERR<n>STATUS.ER is set to 1 when an injected error is recorded.

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

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

CI, bit [8]

Critical Error flag. The value 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 0 when an injected error is recorded.

`1` ERR<n>STATUS.CI is set to 1 when an injected error is recorded.

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

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

CE, bits [7:6]

Corrected Error generation enable. 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 (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 (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 (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 is RES0 if the node does not support this control.

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

DE, bit [5]

Deferred Error generation enable. 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 is RES0 if the node does not support this control.

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

UEO, bit [4]

Latent or Restartable Error generation enable. 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 is RES0 if the node does not support this control.

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

UER, bit [3]

Signaled or Recoverable Error generation enable. 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 is RES0 if the node does not support this control.

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

UEU, bit [2]

Unrecoverable Error generation enable. 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 is RES0 if the node does not support this control.

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

UC, bit [1]

Uncontainable Error generation enable. 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 is RES0 if the node does not support this control.

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

OF, bit [0]

Overflow flag. The value 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 0 when an injected error is recorded.

`1` ERR<n>STATUS.OF is set to 1 when an injected error is recorded.

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

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

Accessibility

None.

4.4.14 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

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.

RES0 otherwise.

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

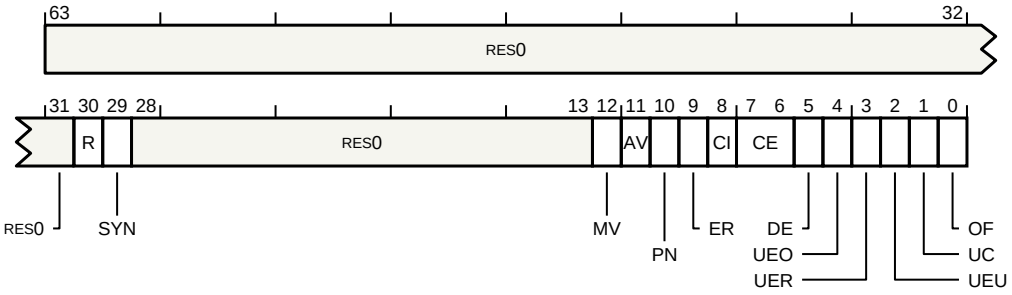
Attributes

When accessed using System registers, 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 memory-mapped registers, ERR<*n*>PFGF are 64-bit read-only registers located at offset 0x800 + 64×*n*.

Field descriptions

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



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 == 0.

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 == 0.

——Note:————————————

If this bit is 1 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 1.

`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 1. ERR<*n*>PGFCTL.MV is RAO.

The syndrome fields that node does not update are unchanged and must be writable when ERR<n>STATUS.MV is set to 0.

——Note:————————————

If this bit is 1 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 1. ERR<n>PFGCTL.AV is RAO.

ERR<n>ADDR must be writable when ERR<n>STATUS.AV is set to 0.

——Note:——————————

If this bit is 1 software can write a specific value into ERR<n>ADDR when setting up a fault injection event.

————————————————

PN, bit [10]

Poison 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 1.

`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.

This bit reads-as-zero if the node does not support this flag.

ER, bit [9]

Error Reported 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.

This bit reads-as-zero if the node does not support this flag.

CI, bit [8]

Critical Error 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 1.

`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.

This bit reads-as-zero if the node does not support this flag.

CE, bits [7:6]

Corrected Error generation. Describes the types of Corrected Error 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 (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 (Corrected Errors that are recorded as ERR<n>STATUS.CE == `0b01` and `0b11`).

All other values are reserved.

This field reads-as-zeros if the node does not support this type of error.

DE, bit [5]

Deferred Error generation. 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.

This bit reads-as-zero if the node does not support this type of error.

UEO, bit [4]

Latent or Restartable Error generation. 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.

This bit reads-as-zero if the node does not support this type of error.

UER, bit [3]

Signaled or Recoverable Error generation. 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.

This bit reads-as-zero if the node does not support this type of error.

UEU, bit [2]

Unrecoverable Error generation. 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.

This bit reads-as-zero if the node does not support this type of error.

UC, bit [1]

Uncontainable Error generation. 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.

This bit reads-as-zero if the node does not support this type of error.

OF, bit [0]

Overflow 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.

This bit reads-as-zero if the node does not support this flag.

Accessibility

None.

4.4.15 ERR<*n*>STATUS, Error Record Primary Status Register

The ERR<*n*>STATUS characteristics are:

Purpose

Contains status information for the error record, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether an error record has been discarded because additional errors have been detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record registers contain valid information.
- Whether the error was 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 the error record 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.

For IMPLEMENTATION DEFINED fields in ERR<*n*>STATUS, writing zero must always be supported to return the error record to an initial 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.

Configurations

Present only if error record *n* is implemented. 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 System registers, ERR<*n*>STATUS is a 64-bit read/write register accessed using:

- `MRC` and `MCR` of ERXSTATUS for ERR<*n*>STATUS[31:0] when ERRSELR.SEL is set to *n*.
- `MRS` and `MSR` of ERXSTATUS_EL1 when ERRSELR_EL1.SEL is set to *n*.

When accessed as memory-mapped registers, ERR<*n*>STATUS are 64-bit read/write registers located at offset `0x010` + 64×*n*.

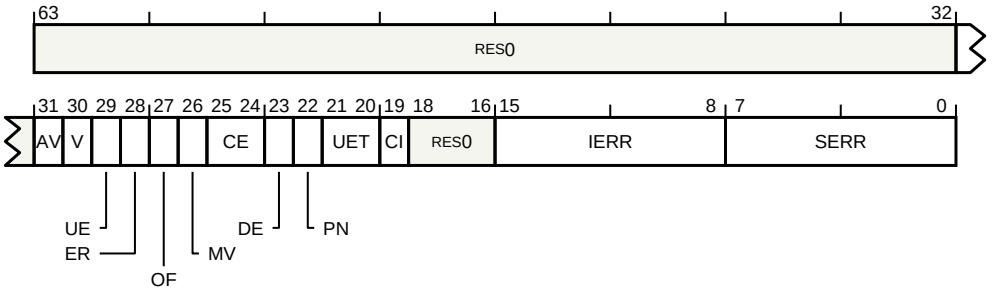
——Note:——————————

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<*q*>CTRL, where *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*.

————————————————

Field descriptions

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



Bits [63:32,18:16]

Reserved. This field is RES0.

AV, bit [31]

Address Valid. The possible values of this bit are:

`0` ERR<n>ADDR not valid.

`1` ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset. This bit is preserved on an Error Recovery reset.

When RAS System Architecture v1.0 is implemented

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

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.

When RAS System Architecture v1.0 is implemented

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and is not being cleared to 0 in the same write.

UE, bit [29]

Uncorrected error. The possible values of this bit are:

`0` No errors have been detected, or all detected errors have been either corrected or deferred.

`1` At least one detected error was not corrected and not deferred.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

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.

When RAS System Architecture v1.0 is implemented

This bit ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write.

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 node to the master making the access or other transaction. This can be because any of the following are true:

- The applicable one of the ERR<*q*>CTLR.{WUE,RUE,UE} bits is implemented and was set to 1 when an Uncorrected error was detected.
- The applicable one of the ERR<*q*>CTLR.{WUE,RUE,UE} bits is not implemented and the node always reports errors.

It is IMPLEMENTATION DEFINED whether this bit can be set to 1 by a Deferred error.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.UE is set to 0 and this bit is never set to 1 by a Deferred error.
- ERR<*n*>STATUS.{UE, DE} are both set to 0 and this bit can be set to 1 by a Deferred error.

This bit is read/write-one-to-clear.

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

When RAS System Architecture v1.1 is implemented

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

When RAS System Architecture v1.0 is implemented

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing ERR<*n*>STATUS.UE to 0, if this bit is never set to 1 by a Deferred error.
- Clearing both ERR<*n*>STATUS.{UE, DE} to 0, if this bit can be set to 1 by a Deferred error.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

——Note:——————————————

An External Abort signaled by the node might be masked and not generate any exception.

————————————————————

OF, bit [27]

Overflow.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

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

When RAS System Architecture v1.1 is implemented

Indicates that multiple errors have been detected. This bit is set to 1 when one of the following occurs:

- A Corrected error counter is implemented, an error is counted, and the counter overflows.
- ERR<*n*>STATUS.V was previously set to 1, a Corrected error counter is not implemented, and a Corrected error is recorded.
- ERR<*n*>STATUS.V was previously set to 1, 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` No error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed, since this bit was last cleared to zero.

`1` At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed, since this bit was last cleared to zero.

When RAS System Architecture v1.0 is implemented

Indicates that multiple errors have been detected. This bit is set to 1 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 1.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and ERR<*n*>STATUS.CE != `0b00`. ERR<*n*>STATUS.CE might be updated for the new Corrected error.
- A Corrected error counter is implemented, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is set to 1 when one of the following occurs:

- A Deferred error is detected and ERR<*n*>STATUS.UE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, and either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 1.
- A Corrected error counter is implemented, either or both the ERR<*n*>STATUS.UE or ERR<*n*>STATUS.DE bits are set to 1, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is cleared to 0 when one of the following occurs:

- An Uncorrected error is detected and ERR<*n*>STATUS.UE == 0.
- A Deferred error is detected, ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 0.
- A Corrected error is detected, ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0 and ERR<*n*>STATUS.CE == `0b00`.

The IMPLEMENTATION DEFINED clearing of this bit might also depend on the value of the other error status bits.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.
- A direct write to this bit that clears this bit to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

The possible values of this bit are:

`0` If ERR<*n*>STATUS.UE == 1, then no error syndrome for an Uncorrected error has been discarded.

If ERR<*n*>STATUS.UE == 0 and ERR<*n*>STATUS.DE == 1, then no error syndrome for a Deferred error has been discarded.

If ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, and a Corrected error counter is implemented, then the counter has not overflowed.

If ERR<*n*>STATUS.UE == 0, ERR<*n*>STATUS.DE == 0, ERR<*n*>STATUS.CE != `0b00`, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded.

——Note:——————————————

This bit might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.

————————————————————

`1` At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

MV, bit [26]

Miscellaneous Registers Valid. 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.

When RAS System Architecture v1.0 is implemented

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

——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.

————————————————————

CE, bits [25:24]

Corrected error. The possible values of this field are:

`0b00` No errors were corrected.

`0b01` At least one transient error was corrected.

`0b10` At least one error was corrected.

`0b11` At least one persistent error was corrected.

The mechanism by which a node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to `0b10` when an error is corrected.

If this field is nonzero, then software must write ones to this field, to clear this field to zero, when clearing ERR<*n*>STATUS.V to 0.

This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

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

When RAS System Architecture v1.0 is implemented

This field ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write.

DE, bit [23]

Deferred error. The possible values of this bit are:

`0` No errors were deferred.

`1` At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

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.

When RAS System Architecture v1.0 is implemented

This bit ignores writes if ERR<*n*>STATUS.OF is set to 1 and is not being cleared to 0 in the same write.

PN, bit [22]

Poison. The possible values of this bit are:

`0` Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC).

——**Note:**——————————

If a producer node detects a corrupt value and defers the error by *producing* a poison value, then this bit is set to 0 at the producer node.

————————————————

`1` Uncorrected error or Deferred error recorded because a poison value was detected.

——**Note:**——————————

This might only be an indication of poison, because, in some EDC schemes, a poison value is encoded as an unlikely form of corrupt data, meaning it is possible to mistake a corrupt value as a poison value.

————————————————

It is IMPLEMENTATION DEFINED whether a node can distinguish a poison value from a corrupt value.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.{DE, UE} are both set to 0.

This bit is read/write-one-to-clear.

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

When RAS System Architecture v1.1 is implemented

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

When RAS System Architecture v1.0 is implemented

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing both ERR<*n*>STATUS.{DE, UE} to 0.

This bit ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

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).

This field is not valid and reads UNKNOWN if any of the following are true:

- ERR<*n*>STATUS.V is set to 0.
- ERR<*n*>STATUS.UE is set to 0.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

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

When RAS System Architecture v1.1 is implemented

If this field is nonzero, then software must write ones to this field, to clear this field to zero, when clearing ERR<*n*>STATUS.V to 0.

When RAS System Architecture v1.0 is implemented

If this field is nonzero, then software must write ones to this field, to clear this field to zero, when any of:

- Clearing ERR<*n*>STATUS.V to 0.
- Clearing ERR<*n*>STATUS.UE to 0.

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

——Note:——————————

Software might use the information in the error record registers to determine what recovery is necessary.

————————————————

CI, bit [19]

Critical error.

When RAS System Architecture v1.1 is implemented

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.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<*n*>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

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.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code SERR value. Further IMPLEMENTATION DEFINED information can be placed in the MISC registers.

This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

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

When RAS System Architecture v1.0 is implemented

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

SERR, bits [7:0]

Architecturally-defined primary error code. Indicates the type of error. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry. The possible values of this field are:

0 No error.

1 IMPLEMENTATION DEFINED error.

2 Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.

3 IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.

4 Assertion failure. For example, consistency failure.

5 Error detected on internal data path. For example, parity on ALU result.

6 Data value from associative memory. For example, ECC error on cache data.

7 Address/control value from associative memory. For example, ECC error on cache tag.

8 Data value from a TLB. For example, ECC error on TLB data.

9 Address/control value from a TLB. For example, ECC error on TLB tag.

10 Data value from producer. For example, parity error on write data bus.

11 Address/control value from producer. For example, parity error on address bus.

12 Data value from (non-associative) external memory. For example, ECC error in SDRAM.

13 Illegal address (software fault). For example, access to unpopulated memory.

14 Illegal access (software fault). For example, byte write to word register.

15 Illegal state (software fault). For example, device not ready.

16 Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, and SIMD&FP registers are data registers.

17 Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, and SIMD&FP registers are control registers.

18 Error response from slave. For example, error response from cache write-back.

19 External timeout. For example, timeout on interaction with another node.

20 Internal timeout. For example, timeout on interface within the node.

21 Deferred error from slave not supported at master. For example, poisoned data received from a slave by a master that cannot defer the error further.

All other values are reserved.

This field is not valid and reads UNKNOWN if ERR<*n*>STATUS.V is set to 0.

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

When RAS System Architecture v1.0 is implemented

This field ignores writes if any of ERR<*n*>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

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 permitted values that can be written to the fields is IMPLEMENTATION DEFINED.

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 error status field is nonzero and 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.

To ensure correct and portable operation, when software is clearing the valid bits in the register to allow new errors to be recorded, software must:

- Determine which fields must be cleared 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.

4.4.16 ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

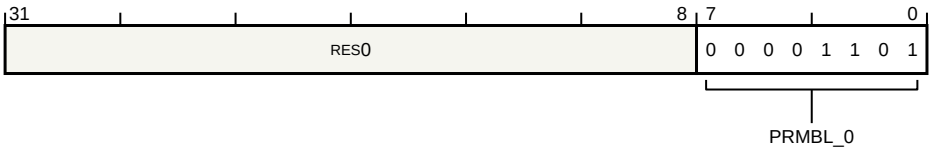
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFF0`.

Field descriptions

The ERRCIDR0 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0. This field reads as `0x0D`.

Accessibility

None.

4.4.17 ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

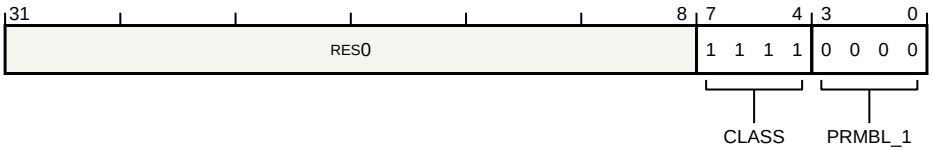
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFF4`.

Field descriptions

The ERRCIDR1 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

CLASS, bits [7:4]

Component class. The defined values of this field are:

`0xF` Generic peripheral 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`.

Accessibility

None.

4.4.18 ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

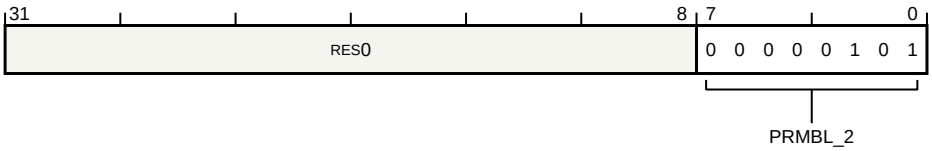
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFF8`.

Field descriptions

The ERRCIDR2 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2. This field reads as `0x05`.

Accessibility

None.

4.4.19 ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

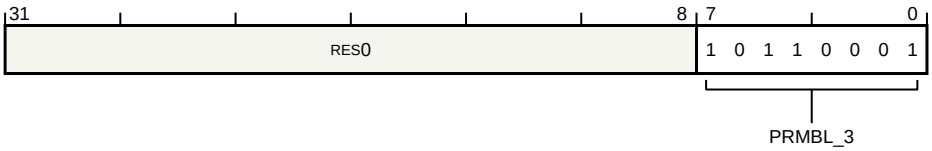
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRCIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFFC`.

Field descriptions

The ERRCIDR3 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3. This field reads as `0xB1`.

Accessibility

None.

4.4.20 ERRDEVAFF, Device Affinity Register

The ERRDEVAFF characteristics are:

Purpose

For a group of error records that has affinity with a single PE or cluster 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 cluster 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 affinity level is 2 then all of the following are true:

- The group of error records has affinity with all PEs in the system that have the same value in MPIDR_EL1.{Aff3,Aff2}.
- ERRDEVAFF.{Aff3,Aff2} reads the same value as MPIDR_EL1.{Aff3,Aff2}.
- ERRDEVAFF.{Aff1} reads as `0x80` and ERRDEVAFF.{Aff0,F0V} read-as-zero, to indicate affinity level 2.

Configurations

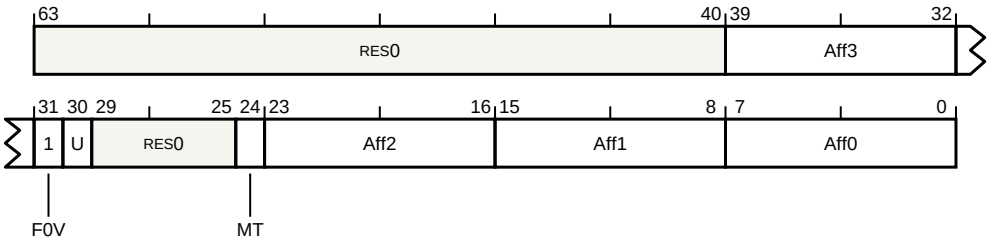
Present only if a group of error records has affinity with a PE or cluster of PEs. RES0 otherwise.

Attributes

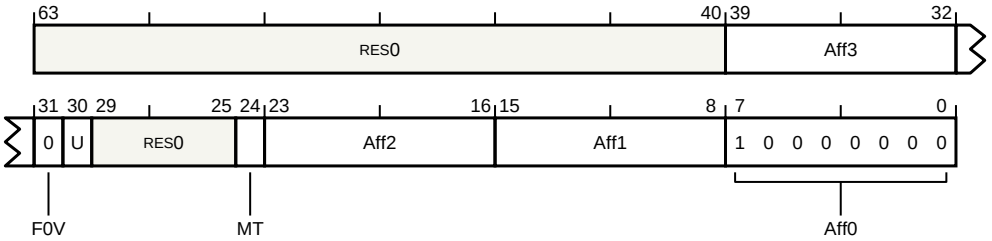
ERRDEVAFF is a 64-bit read-only memory-mapped register located at offset `0xFA8`.

Field descriptions

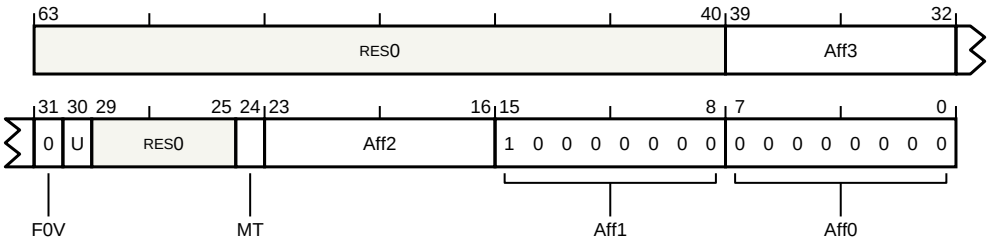
When level 0, the ERRDEVAFF bit assignments are:



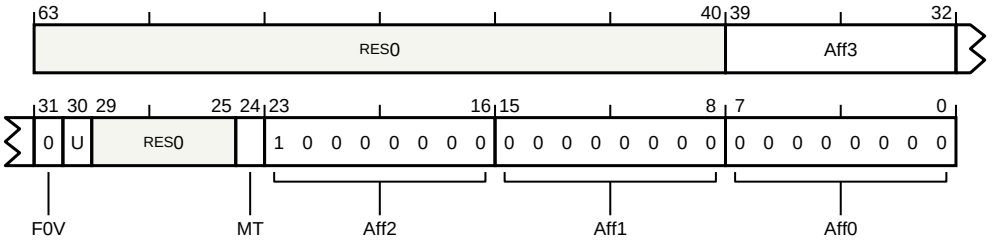
When level 1, the ERRDEVAFF bit assignments are:



When level 2, the ERRDEVAFF bit assignments are:



When level 3, the ERRDEVAFF bit assignments are:



Bits [63:40,29:25]

Reserved. This field is RES0.

Aff3, bits [39:32]

Affinity level 3. The MPIDR_EL1.Aff3 field viewed from the highest Exception level of the associated PE or PEs.

F0V, bit [31]

Indicates the Aff0 bit is valid. The defined values of this bit are:

`0b0` Aff0 is not valid.

`0b1` Aff0 is valid.

U, bit [30]

Uniprocessor. The MPIDR_EL1.U bit viewed from the highest Exception level of the associated PE.

This bit is not valid and reads UNKNOWN if ERRDEVAFF.Aff0 is not valid.

MT, bit [24]

Multithreaded. The MPIDR_EL1.MT bit viewed from the highest Exception level of the associated PE.

This bit is not valid and reads UNKNOWN if ERRDEVAFF.Aff0 is not valid.

Aff2, bits [23:16], when level 0, level 1, or level 2

Affinity level 2. The MPIDR_EL1.Aff2 field viewed from the highest Exception level of the associated PE or PEs.

Aff2, bits [23:16], when level 3

Indicates whether the Aff3 field is valid. The defined values of this field are:

`0x80` Aff3 is valid.

All other values are reserved.

This field reads as `0x80`.

Aff1, bits [15:8], when level 0, or level 1

Affinity level 1. The MPIDR_EL1.Aff1 field viewed from the highest Exception level of the associated PE or PEs.

Aff1, bits [15:8], when level 2, or level 3

Indicates the Aff2 field is valid. The defined values of this field are:

`0x80` Aff2 is valid.

`0x00` Aff2 is not valid.

All other values are reserved.

Aff0, bits [7:0], when level 0

Affinity level 0. The MPIDR_EL1.Aff0 field viewed from the highest Exception level of the associated PE or PEs.

Aff0, bits [7:0], when level 1, level 2, or level 3

Indicates the Aff1 field is not valid. The defined values of this field are:

`0x80` Aff1 is valid.

`0x00` Aff1 is not valid.

All other values are reserved.

Accessibility

None.

4.4.21 ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

Configurations

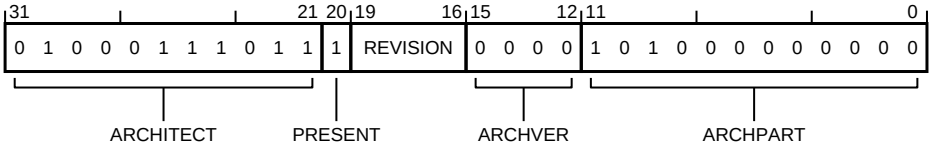
Always implemented.

Attributes

ERRDEVARCH is a 32-bit read-only memory-mapped register located at offset `0xFBC`.

Field descriptions

The ERRDEVARCH bit assignments are:



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`.

Accessibility

None.

4.4.22 ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

Purpose

Provides discovery information for the component.

Configurations

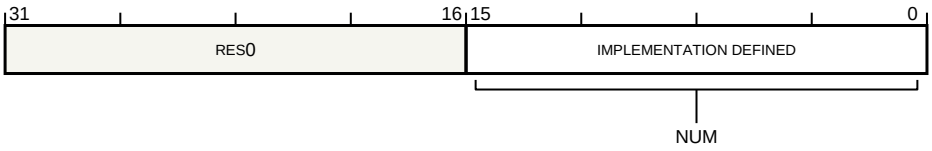
Always implemented.

Attributes

ERRDEVID is a 32-bit read-only memory-mapped register located at offset `0xFC8`.

Field descriptions

The ERRDEVID bit assignments are:



Bits [31:16]

Reserved. This field is RES0.

NUM, bits [15:0]

Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This manual describes the memory-mapped view of a group with up to 56 records, the most that can be contained in a 4KB component, meaning the highest possible value for this field is 56.

This field reads as an IMPLEMENTATION DEFINED value.

Accessibility

None.

4.4.23 ERRGSR, Error Group Status Register

The ERRGSR characteristics are:

Purpose

Shows the status for the records in the group.

Configurations

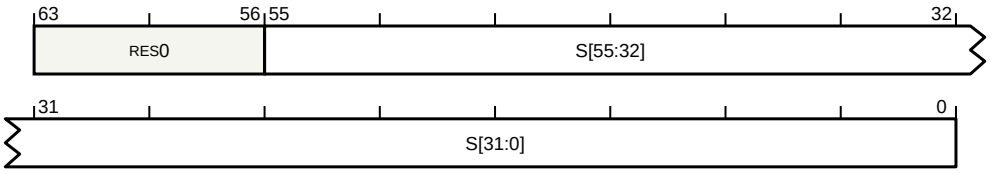
This manual describes the memory-mapped view of a group with up to 56 records, the most that can be contained in a 4KB component. Extra records might be added by increasing the page size and extending ERRGSR into multiple registers.

Attributes

ERRGSR is a 64-bit read-only memory-mapped register located at offset `0xE00`.

Field descriptions

The ERRGSR bit assignments are:



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. The defined values of this bit are:

`0` No error.

`1` One or more errors.

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

- The record is not implemented.
- The record does not support this type of reporting.

Accessibility

None.

4.4.24 ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

Purpose

Defines the implementer of the component.

Configurations

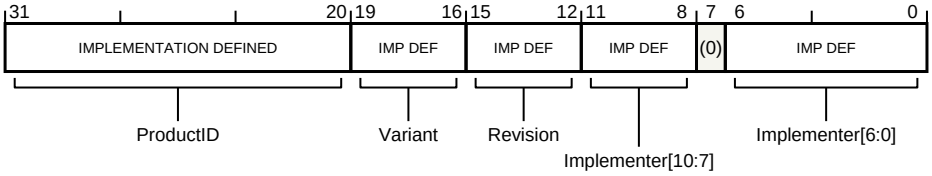
Present only if IMPLEMENTATION DEFINED. RES0 otherwise.

Attributes

ERRIIDR is a 32-bit read-only memory-mapped register located at offset `0xE10`.

Field descriptions

The ERRIIDR bit assignments are:



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 must be 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 0.

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 must be increased to ensure that software can differentiate the different revisions of the component.
  - If Variant is increased then Revision should be set to 0.
- Otherwise, ERRIIDR.Revision must be 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.

Accessibility

None.

4.4.25 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.

Configurations

Present only if the interrupt configuration registers are implemented. RES0 otherwise.

Attributes

ERRIRQCR<*n*> are 64-bit read/write memory-mapped registers located at offset `0xE80` + 8×*n*.

Preface

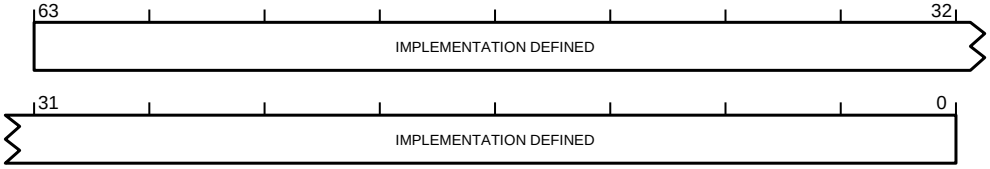
The architecture provides a recommended layout for the ERRIRQCR<*n*> registers. These registers are named ERRFHICR<0-2>, ERRERICR<0-2>, and ERRCRICR<0-2>, for the fault-handling, error-recovery, and critical error interrupts respectively, and ERRIRQSR:

- ERRFHICR<*m*> map to ERRIRQCR0 and ERRIRQCR1.
- ERRERICR<*m*> map to ERRIRQCR2 and ERRIRQCR3.
- ERRCRICR<*m*> map to ERRIRQCR4 and ERRIRQCR5.
- ERRIRQSR maps to ERRIRQCR15.

See ERR<irq>CR0, ERR<irq>CR1, ERR<irq>CR2, and ERRIRQSR. This section describes the generic, IMPLEMENTATION DEFINED, format.

Field descriptions

The ERRIRQCR<0-15> bit assignments are:



Bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

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

Accessibility

None.

4.4.26 ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose

Interrupt status register.

Configurations

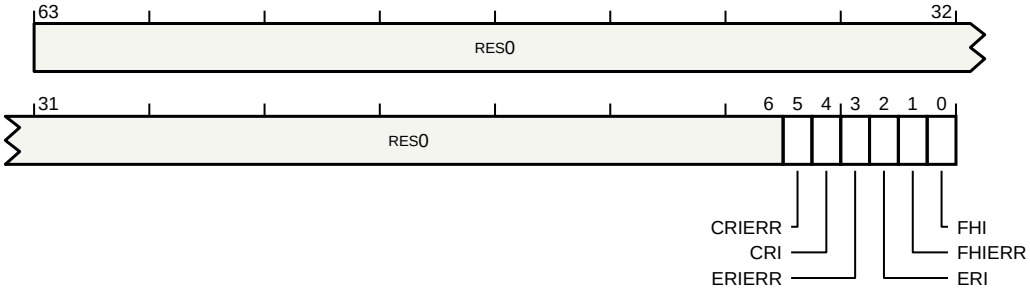
Present only if interrupt configuration registers use the recommended format. RES0 otherwise.

Attributes

ERRIRQSR is a 64-bit read/write memory-mapped register located at offset `0xEF8`.

Field descriptions

The ERRIRQSR bit assignments are:



Bits [63:6]

Reserved. This field is RES0.

CRIERR, bit [5]

Critical Error Interrupt error.

When RAS System Architecture v1.1 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 RAS System Architecture v1.1 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. 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.

ERI, bit [2]

Error Recovery Interrupt write in progress. 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.

————————————————

FHIERR, bit [1]

Fault Handling Interrupt error. 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.

FHI, bit [0]

Fault Handling Interrupt write in progress. 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.

————————————————

Accessibility

None.

4.4.27 ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

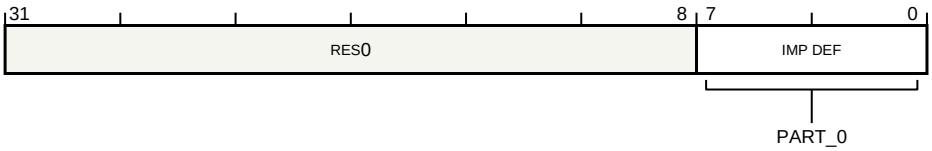
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRPIDR0 is a 32-bit read-only memory-mapped register located at offset `0xFE0`.

Field descriptions

The ERRPIDR0 bit assignments are:



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 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 available to define the revision of the component.

This field reads as an IMPLEMENTATION DEFINED value.

Accessibility

None.

4.4.28 ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

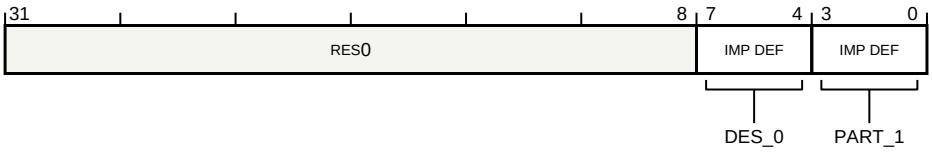
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRPIDR1 is a 32-bit read-only memory-mapped register located at offset `0xFE4`.

Field descriptions

The ERRPIDR1 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. 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.

Note that for a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field reads as an IMPLEMENTATION DEFINED value.

PART_1, bits [3:0]

Part number, bits [11:8]. This field reads as an IMPLEMENTATION DEFINED value.

Accessibility

None.

4.4.29 ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

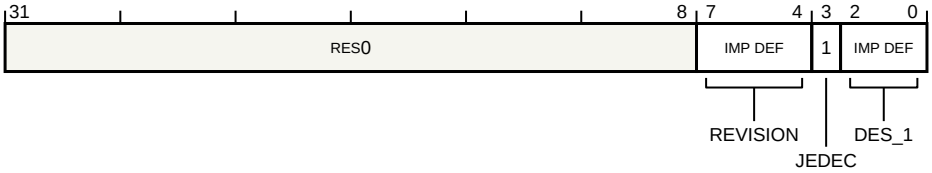
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

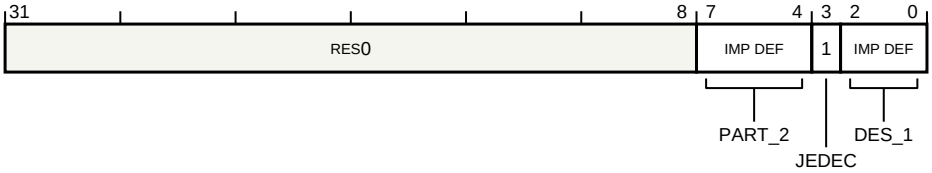
ERRPIDR2 is a 32-bit read-only memory-mapped register located at offset `0xFE8`.

Field descriptions

When 12-bit part number, the ERRPIDR2 bit assignments are:



When 16-bit part number, the ERRPIDR2 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

REVISION, bits [7:4], when 12-bit part number

Component major revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part. When a component is changed, REVISION or REVAND must be increased to ensure that software can differentiate the different revisions of the component. If REVISION is increased then REVAND should be set to 0.

This field reads as an IMPLEMENTATION DEFINED value.

PART_2, bits [7:4], when 16-bit part number

Part number, bits [15:12]. This field reads as an IMPLEMENTATION DEFINED value.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used. This bit reads as one.

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.

Note that for a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field reads as an IMPLEMENTATION DEFINED value.

Accessibility

None.

4.4.30 ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

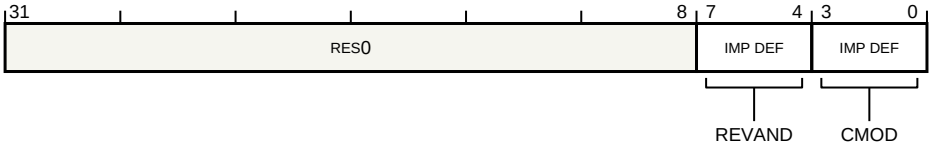
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

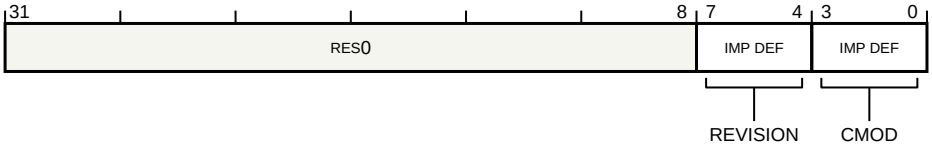
ERRPIDR3 is a 32-bit read-only memory-mapped register located at offset `0xFEC`.

Field descriptions

When 12-bit part number, the ERRPIDR3 bit assignments are:



When 16-bit part number, the ERRPIDR3 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

REVAND, bits [7:4], when 12-bit part number

Component minor revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part. When a component is changed, REVISION or REVAND must be increased to ensure that software can differentiate the different revisions of the component. If REVISION is increased then REVAND should be set to 0.

This field reads as an IMPLEMENTATION DEFINED value.

REVISION, bits [7:4], when 16-bit part number

Component revision. When a component is changed, REVISION must be 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.

Accessibility

None.

4.4.31 ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose

Provides discovery information for the component.

Configurations

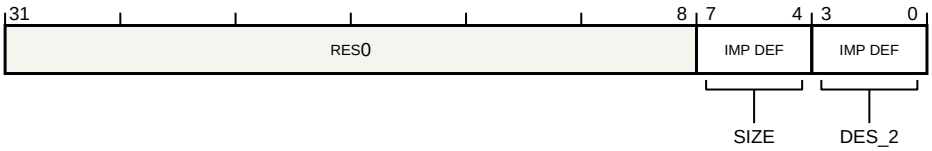
Present only if Peripheral Identification scheme is implemented. RES0 otherwise.

Attributes

ERRPIDR4 is a 32-bit read-only memory-mapped register located at offset `0xFD0`.

Field descriptions

The ERRPIDR4 bit assignments are:



Bits [31:8]

Reserved. This field is RES0.

SIZE, bits [7:4]

Size of the component.

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^{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.

Note that for a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

This field reads as an IMPLEMENTATION DEFINED value.

Accessibility

None.

5 Appendix – Glossary

Asynchronous exception

In the Armv8 architecture, an exception for which any of the following apply:

- The exception is not generated as a result of direct execution or attempted execution of the instruction stream.
- The return address presented to the exception handler is not guaranteed to indicate the instruction that caused the exception.
- The exception is imprecise.

Asynchronous exceptions are also known as interrupts.

Availability

Readiness for correct service.

Baseboard Management Controller

A PE dedicated to system control and monitoring.

BIST *Built-in self-test*

Built-in self-test

A mechanism that permits a machine to test itself.

Catastrophic failure

A failure with harmful consequences that are orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery.

CE *Correctable or Corrected Error*

Contained or containable error

An error that is not uncontained or uncontainable.

Containment

Limiting or preventing the silent propagation of an error. Arm recommends that the scope to which an error is contained is specified.

Correctable or Corrected Error

An error that is detected by hardware and that hardware can correct.

DECTED

Double error correct, triple error detect EDAC. This can detect a single, double or triple bit error and correct a single or double bit error in a protection granule.

Deferred error

An error that has not been silently propagated but does not require immediate action at the producer. The error might have passed from the producer to a consumer.

Detected error

An error that has been detected and signaled to a consumer.

Detected Uncorrectable Error

A detected error that cannot be corrected and causes failure.

Device memory

Memory locations where an access to the location can cause side-effects, or where the value returned for a load can vary depending on the number of loads performed. Typically, the Device memory attributes are used for memory-mapped peripherals and similar locations.

Double fault

A second error that is detected when the PE is in the process of handling a first error condition.

DUE *Detected Uncorrectable Error*

DUE FIT rate

The FIT rate for failures from a DUE.

ECC *Error Correction Code*

EDAC *Error Detection and Correction Code*

EDC *Error Detection Code*

Error

Deviation from correct service or a correct value.

Error Correction Code or Error Detection and Correction Code

A code capable of detecting and correcting a number of errors.

Error Detection Code

A code capable of detecting, but not correcting, errors.

Error propagation

Passing an error from a producer to a consumer.

Error record

Data recorded about an error, usually by hardware.

Error log

Historical data recorded about errors, usually by software.

Error synchronization event

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

Exception

An exception handles an event. For example, an exception could handle an external interrupt or an undefined instruction.

External abort

An in-band error that is generated as a response to a transaction. The name derives from the specific case of an abort generated by a memory system that is external to a PE, but the concept can apply to other interfaces.

In the context of the Arm architecture, this also refers to the type of exception generated when consuming an external abort.

Fail-safe

A failure mode in which the PE and other system components switch to backup mechanisms that keep processing instructions and data to allow either a safe shutdown or restart of the system, or to continue processing critical functions, or both.

Fail-secure

A failure mode in which the PE and other system components fail but the system is secured to allow either a safe shutdown or restart of the system, or to continue processing critical functions without exposing secret data, or both.

Fail-signaled

A failure mode in which the PE signals to the system that it has failed. It might continue to process instructions, but the system must ignore its output, or treat all outputs as detected errors.

Fail-silent

Failure mode in which the PE and all other system components (such as DMAs) stop processing instructions. A watchdog process will detect the failure and restart the system with an Error Recovery reset.

Failure

The event of deviation from correct service.

Failure In Time

The number of expected failures per billion hours of operation.

Fault

The cause of an error.

Fault injection

The deliberate injection of faults into a system for testing.

Fault prevention

Designing a system to avoid faults.

Fault removal

Logic or other mechanisms for detecting faults and correcting or bypassing their effect.

Field Replaceable Unit

The smallest unit that can be replaced without return to base.

FIT *Failure In Time*

FRU *Field Replaceable Unit*

General-purpose registers

The registers that the base instructions use for processing:

- In AArch32 state the general-purpose registers are R0-R14.
- In AArch64 state the general-purpose registers are R0-R30.

Generic Interrupt Controller

Arm system architecture interrupt controller for IRQ and FIQ interrupt exceptions.

GIC *Generic Interrupt Controller*

Hardware fault

A fault that originates in, or affects, hardware.

Imprecise exception

An exception that is not precise.

Infected

Being in error.

Interrupt

In a PE context, an asynchronous exception. There are three interrupt exceptions: IRQ, FIQ and SError. IRQ and FIQ are always precise.

In a system architecture context, an asynchronous event sent to a PE or GIC for processing as an interrupt exception.

Isolation

Limiting the impact of an error only to components that actually try to use corrupted data.

Latent fault

An error that is present in a system but not yet detected.

MBIST

Memory BIST.

Minor failure

A failure with harmful consequences that are of a similar cost to the benefits that are provided by correct service delivery.

Normal memory

Used for bulk memory operations. Hardware might speculatively read these locations.

PE *Processing element*

Persistent fault

A fault that is not transient.

PFA *Predictive Failure Analysis*

Poisoned

State that has been marked as being in error so that subsequent consumption of the state will signal a detected error to a consumer.

Precise exception

An exception where the exception handler receives the state of the PE and the state of the memory system consistent with the PE having executed all of the instructions up to, but not including, the point in the instruction stream where the exception was taken. The state of the PE and the state of the memory do not include instructions that occurred after this point.

Predictive Failure Analysis

Mechanisms to analyze errors and predict future failures.

Processing element (PE)

The abstract machine defined in the Armv8 architecture, as documented in an Arm Architecture Reference Manual. A PE implementation compliant with the Armv8 architecture must conform with the behaviors described in the corresponding Arm Architecture Reference Manual.

Protection granule

A quantum of memory for which an EDC or ECC provides detection or correction. For example, a 72/64 SECDED ECC scheme has a 64-bit protection granule.

RAS *Reliability, Availability, Serviceability*

Recoverable error

A contained error that must be corrected to allow the correct operation of the system or smaller parts of the system to continue.

Reliability

Continuity of correct service.

Restartable error

A contained error that does not immediately impact correct operation. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part.

SDC *Silent Data Corruption*

SDC FIT rate

The FIT rate for failures because of SDC.

SDEC

Single device error correction EDAC. This can detect and correct multiple clustered errors in a protection granule, such as the types of errors that might be seen if a protection granule is striped across multiple devices and multiple errors come from a single device.

SECDED

Single error correct, double error detect EDAC. This can detect a single or double bit error and correct a single bit error in a protection granule.

SED

Single error detect EDC. This can detect a single bit error in a protection granule.

SError Interrupt

An asynchronous interrupt in the Armv8 architecture.

Serviceability

The ability to undergo modifications and repairs.

Service failure mode

A mode entered to reduce the severity of an error.

Silent Data Corruption

An error that is not detected by hardware or software.

Silently propagated

An error that is passed from place to place without being signaled as a detected error.

Software fault

A fault that originates in and affects software.

System Control Processor

A PE dedicated to system control and monitoring.

Synchronous exception

In the Armv8 architecture, an exception for which all of the following apply:

- The exception is generated as a result of direct execution or attempted execution of an instruction.
- The return address presented to the exception handler is guaranteed to indicate the instruction that caused the exception.
- The exception is precise.

Synchronous External Abort

A synchronous exception in the Armv8 architecture.

Transient fault

A fault that is not persistent.

Uncontained or uncontainable error

An error that has been, or might have been, silently propagated.

Undetected fault

See *Latent fault*.

Unrecoverable error

A contained error that is not recoverable. Continued correct operation is generally not possible. Usually this means correct operation of the system, but it can also be used in other contexts to describe correct operation of a smaller part. Systems might use high-level recovery techniques to work around an unrecoverable yet contained error in a component so that the system recovers from the error.