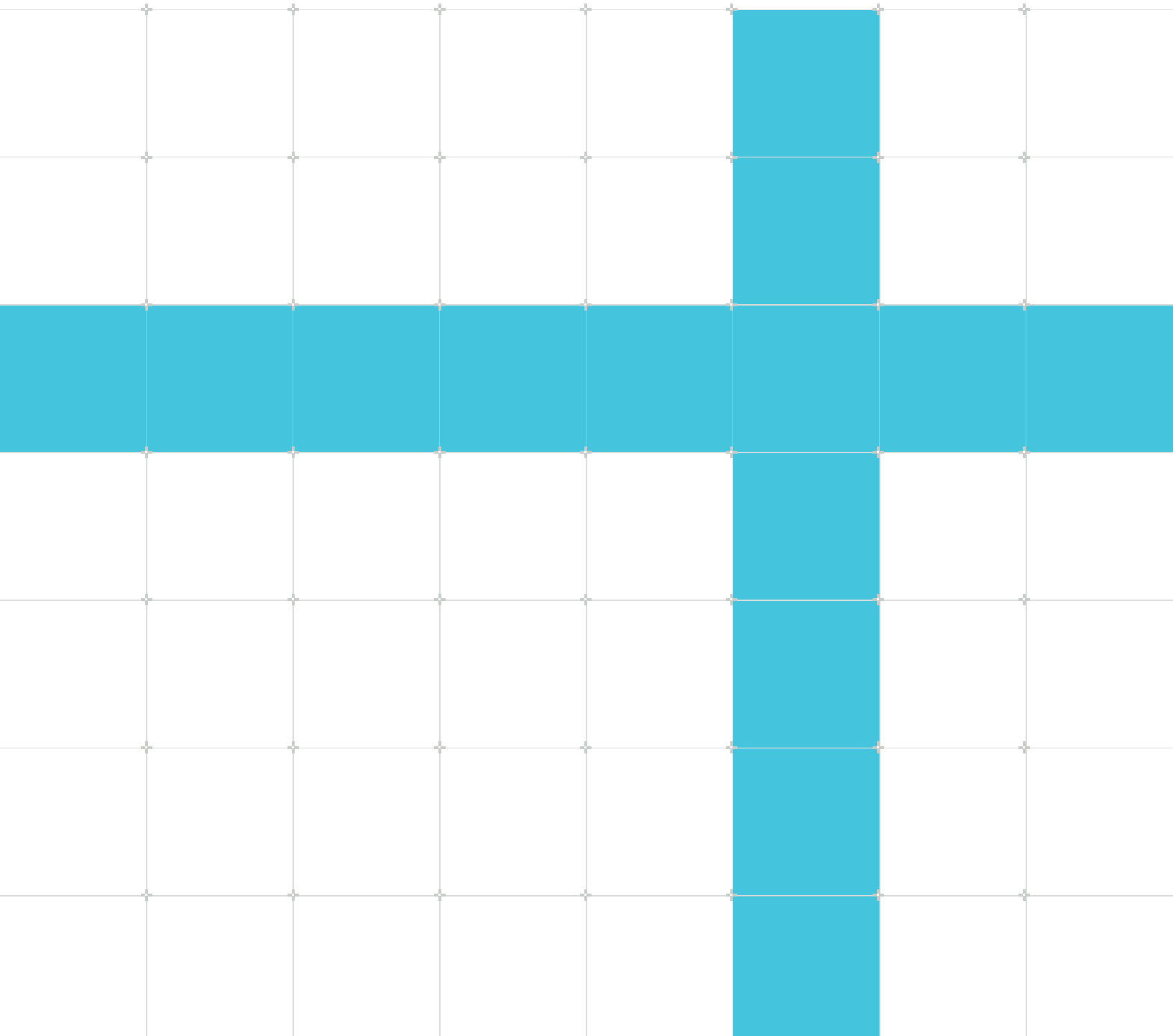
Learn the architecture - RAS Hardware Guide

Version 1.0

Issue 00
110247_100_00_en



Learn the architecture - RAS Hardware Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-00	12 August 2025	Non-Confidential	First release

Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm

makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

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

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

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

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

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction to RAS

Reliability, Availability, and Serviceability (RAS) are crucial attributes in modern computing architectures, ensuring system robustness and operational continuity. In hardware systems, RAS hardware features are implemented to detect, report, and recover from faults. RAS features enhance overall system dependability.

The Arm RAS System Architecture specification defines components such as Nodes and Error Records, along with terminology for error handling and containment. The RAS Overview guide helps explain the terms used in this guide.

1.1 Importance of RAS

RAS is important for:

Enhancing reliability

Reliability refers to the ability to function correctly over time without failures. We will describe the importance of reliability in Arm IPs and the measures taken to enhance it, such as error detection mechanisms.

Ensuring availability

Availability measures a system's readiness to deliver correct service when required. This document describes mechanisms in Arm A-class systems that enhance availability, such as error containment and support for fault recovery policies.

Facilitating serviceability

Serviceability refers to the ease with which faults can be diagnosed, isolated, and resolved. This document emphasizes the role of standard error records, RAS registers, and fault handling agents in supporting modular, diagnosable, and maintainable Arm-based IPs.

1.2 RAS mechanisms in Arm hardware

Arm hardware implements following mechanisms:

Error detection techniques

Error detection is fundamental to RAS and relies on various mechanisms. Parity checking detects single-bit errors using an additional parity bit. Error Correcting Code (ECC) detects and corrects single-bit errors and can detect double bit errors.

Error recording and reporting

Errors are logged in standard error records that store details such as:

1. Error code: Identifies the nature of the error
2. Associated memory address: Indicates the error location
3. Counter: diagnosing recurring errors and reduce the impact of fault handling

4. Interrupt mechanisms: Includes Fault Handling Interrupts (FHIs), Error Recovery Interrupts (ERIs), and Critical Error Interrupts (CRIs)

Error injection

Pseudo fault generation also known as error injection validates RAS mechanisms. Hardware-Assisted Error Injection uses dedicated registers to simulate errors.

2. CPU RAS overview

This chapter describes the RAS features, error detection mechanisms, error reporting infrastructure, and fault recovery strategies implemented in CPUs.

Reliability, Availability, and Serviceability (RAS) are essential in modern computing architectures, particularly in high-performance Arm A-Class CPUs. These capabilities enable systems to detect, isolate, and recover from hardware errors, enhancing stability and reducing downtime. The Arm RAS architecture provides a standardized framework for error management, making it well-suited for enterprise, cloud, and embedded deployments. This chapter describes the key RAS features in Arm CPUs, including error detection, reporting infrastructure, and fault recovery strategies.

2.1 Memory and cache error detection mechanisms

CPUs incorporate multiple mechanisms to detect faults in memory, caches, execution units, and interconnects. These mechanisms ensure that faults are identified before they can cause critical failures. The mechanisms are:

- Error Correcting Code (ECC): Protects L1, L2, and L3 caches, system memory, and interconnect buffers. It uses SECDED (Single Error Correction and Double Error Detection) ECC to correct single-bit errors and detect multi-bit errors.
- Parity Checking: Implemented in instruction caches, Translation Lookaside Buffers (TLBs), and branch predictors detect single-bit errors.

2.2 Error containment and poisoning

Error containment in Arm CPUs is a critical component of the RAS architecture, ensuring that faults are detected and isolated before they can propagate and affect system stability. One of the key techniques used for error containment is data poisoning, which marks corrupted or uncorrectable data to prevent it from being consumed by the processor.

When an uncorrectable error is detected in a cache or memory location, the affected data is tagged as poison. This ensures that any subsequent access to the poisoned data results in a fault or an error notification, triggering appropriate system-level handling. For example, in L1 and L2 caches, a single-bit correctable error via ECC allows normal operation. But if a double-bit uncorrectable error is found, the cache line is poisoned and upon consumption of the poisoned data, a data abort exception is triggered, preventing erroneous execution. Also, poisoned data can propagate through the AMBA CHI/AXI interconnect, ensuring that other cores or devices accessing that data are notified of the issue and can take corrective action.

To further contain errors, data poisoning ensures that memory transactions involving corrupted data are flagged before reaching execution stages. This allows software or firmware to discard the faulty data or initiate a recovery process. In multi-core environments, where data is shared between

processors, poison tracking prevents one core from consuming data marked as poisoned by another core. Enhancing overall system resilience.

An example of error containment in action is when an uncorrectable ECC error is detected in the L3 cache of a multi-core Arm server. The cache marks the affected line as poisoned, which is recorded and signaled via a Fault Handling Interrupt (FHI). The cache line that is marked poisoned propagates this status via the interconnect. If another core attempts to read the poisoned cache line, the poison is propagated to the other core's cache. If the core tries to consume the poisoned data, an exception, such as a Synchronous External Abort (SEA), is raised, preventing execution with faulty data. Depending on system policy, the core might recover using redundancy techniques or escalate the issue through error reporting mechanisms. Importantly, poisoning also enables software-level recovery: when the offending instruction is aborted, the operating system can isolate and terminate only the affected process. This enables the rest of the system to continue operating normally, preserving overall system availability and minimizing service disruption.

By implementing these containment strategies, Arm CPUs ensure that faults do not silently corrupt computations or destabilize the system, maintaining robust reliability for enterprise, cloud, and embedded applications.

2.3 Error recording in Arm CPUs

Error recording in CPUs is a fundamental component of the RAS framework which enables reliable fault detection and efficient communication to software and system managers. Errors are categorized by priority. The assignment of priority to interrupt type is configurable. Arm recommend the following, but it is not mandatory:

- Corrected error (CE) generates FHI
- Deferred error (DE) generates FHI
- Uncorrected error (UE) generates ERI

In cases of critical error conditions, CRIs are generated.

Errors are logged in RAS error records, ERR<n>STATUS, which capture

- Whether an error occurred
- Its severity (classified as correctable, uncorrectable, or deferred)
- Whether the error was reported, overflowed, or consumed
- Whether the associated error record data is valid.

The register also flags whether the error was reported because poison data was detected or because a corrupt value was detected by an error detection code. This standardized format enables for consistent logging and software handling across components that implement the RAS architecture.

CEs are often tracked via a dedicated counter. To avoid excessive fault-handling overhead, Arm supports error thresholding, where interrupts are generated only after a specified number of CEs.

This is especially useful in scenarios such as stuck-at faults, where each read could trigger a CE. Instead of interrupting on every CE, the system can be configured to raise an interrupt after every n errors by programming the counter in the thresholding registers. This approach balances system performance with fault visibility. When the fault condition is mitigated, for example by isolating the faulty location, software can reset the thresholding registers to resume finer-grained error reporting. These mechanisms ensure that software, hypervisors, and firmware can take timely corrective actions, such as retrying operations, isolating faulty components, or performing a system reset, thereby preserving system stability and availability.

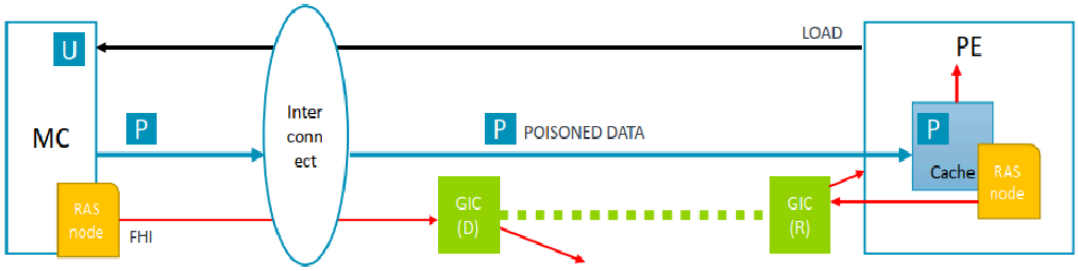
2.4 RAS reporting mechanisms

RAS interrupts provide a standardized mechanism for the hardware to communicate error events to software, enabling timely and appropriate fault handling. Arm A-Class CPUs support a range of reporting mechanisms, each targeting a specific classes of error severity and system response requirement. The hardware enables configurability over which detected errors trigger which interrupts, except for some error types that follow fixed signalling paths. There are two sets of reporting mechanisms:

- When the error is detected and recorded in a RAS error record.
  - ERI: Used for UEs, ERI provides software with the opportunity to act before the error escalates. Deferred errors can also be signaled by ERI, but use of FHI is more likely and is recommended.
  - FHI: Typically used for CEs and DEs. It is the preferred interrupt for errors that do not impact current execution but might indicate underlying hardware degradation or patterns of concern.
  - CRI: A platform-defined signal typically reserved for severe conditions that require immediate system-level intervention. Its implementation and use are system specific.
- When the error is consumed by a CPU.
  - SEA: An exception that occurs immediately when an UE impacts program execution. Because a UE directly affects program behavior, it must be handled by the operational software managing the executing process.
  - System Error exception (SError): An asynchronous exception used to report errors that may not be directly related to the instruction currently executing.

SEA and SError are not RAS interrupts. They are the way a CPU responds to an in-band error response when it tries to consume the error. In contrast, ERI, FHI, and CRI are interrupts that signal that the detection of an error has resulted in the error being recorded in an error record. The following diagram shows an ideal memory error flow example of when there is uncorrected error on read.

Figure 2-1: Example memory error flow example



1. CPU (PE) issues a load from memory location, misses in the cache.
2. Memory Controller (MC) reads memory location and detects an uncorrectable error.
3. The read is completed, and poison is returned for the corrupt location.
4. CPU receives poisoned data, stores it internally for example in the cache.
5. MC updates RAS node with DE information and generates an FHI. Note that the interrupt receiver is not always the final handler or the requester that consumes the error. For example, a first application processor (AP) might access memory and trigger detection of an uncorrectable error, which the system defers using poison. The system records the error and issues an FHI or ERI (or both), while the first AP later consumes the poison and raises an SEA or SEI exception. The FHI or ERI may then be handled by a different AP or by the SatMC.
6. CPU commits to executing load instruction consumes erroneous data, reads poisoned data from the cache or buffer.
7. CPU then generates a synchronous external abort.

The choice of reporting an error consumed by the CPU as either an SEA or SError is typically not software configurable, and might depend on the type or error and/or memory being accessed. For example, a common design choice for high performance Arm server CPUs is to report errors on reads of Normal memory using SEA, and errors on reads of Device memory as SError. This is done to balance performance and reliability.

3. RAS Within the Wider System

RAS is a system-wide concern. While the CPU is central to RAS operations, many other IP blocks in an SoC also play a role in error detection, containment, and recovery. This chapter describes how key components, such as interconnects, memory management units, interrupt controllers, and memory subsystems, implement RAS features, using a typical Arm-based compute subsystem as a reference point.

3.1 RAS in the interconnect

The system interconnect is responsible for data movement between cores, memory, and peripherals. It also includes several reliability features:

- Internal ECC/parity protection: Internal RAMs, such as cache memory and snoop filters, are typically protected with Single Error Correction, Double Error Detection (SECDED) ECC. This helps detect and correct transient or permanent bit-flips in control or data paths.
- Error classification: Error classification into CEs, DEs, and UEs.
- Error logging and reporting: Each component in the interconnect can implement local error records with standardized fields for status, address, syndrome, and metadata. These are accessible via memory-mapped registers.
- Interrupt signalling: Correctable and uncorrectable errors can be signaled to platform controllers or firmware via dedicated fault handling or recovery interrupt lines.
- Poison propagation: Errors in data are often tracked and tagged as “poisoned,” and this tag is propagated through the system until the data is either consumed, triggering an exception, or overwritten.
- Pseudo-fault injection: Many interconnects support fault simulation, enabling software to inject synthetic errors for validation and stress-testing RAS handling flows.

3.2 RAS in the system memory management unit

The System Memory Management Unit (SMMU) subsystem performs address translations and plays a key role in enforcing memory protection policies. It can also include robust RAS support:

- Protected structures: Components such as Translation Lookaside Buffers (TLBs), configuration caches, and metadata tables are often protected by ECC to ensure integrity during address translation.
- Error detection and containment: Errors encountered during translation table walks, or in configuration structures, are classified and handled appropriately. The SMMU might support deferral of recoverable faults and logging of unrecoverable ones.
- Standard error records: MMUs typically implement RAS-compliant error records that capture the fault type, affected address, and other syndrome information. These support centralized fault reporting frameworks.

- Interrupt generation: MMUs can generate interrupts in response to various error conditions. Such as when errors are correctable, deferred for later handling, or uncorrected but potentially recoverable or critical to system stability.
- Error injection: Testing registers enable software to simulate faults in translation logic or access control for validation of error-handling software.

3.3 RAS in the interrupt controller

The interrupt controller is essential for timely error notification and typically includes dedicated RAS support to ensure its internal state and memory are dependable:

- ECC protection: Internal RAMs used for interrupt state, configuration, or routing metadata are usually protected against single and multi-bit errors.
- Error detection and logging: Interrupt controllers implement error status registers and logs for internal faults, including parity or ECC errors in protected RAMs.
- Error interrupts: Standard RAS interrupts can be generated on detection of faults. For example, Correctable errors can raise a fault handling interrupt and more severe errors may raise an error recovery or critical interrupt.
- Scrubbing support: Some designs include scrubbing capabilities to periodically scan internal memories for latent faults and proactively detect issues before they affect operation.
- Error injection: Error injection in the interrupt controller allows software to deliberately introduce ECC faults into internal memories to validate error detection, logging, and recovery mechanisms.

3.4 System-level coordination

To support scalable RAS implementation, many components conform to a standardized error reporting model, with features such as:

- Uniform error records: Common formatting across components simplifies log correlation and analysis.
- Interrupt routing: Error interrupts from various blocks can be routed to dedicated handlers in platform firmware or a control processor.
- Centralized access via utility bus: A system-wide utility bus often provides memory-mapped access to error records and control registers across subsystems.
- Error injection: Many components include control registers for injecting test errors in a safe, controlled manner.

These example features ensure that systems can detect faults early, isolate them from propagating, and provide software with the necessary visibility and tools for corrective action. This cross-IP coordination forms the backbone of system-level reliability and supports use cases ranging from enterprise compute to safety-critical applications.

4. RAS system control and recording

This chapter describes the mechanisms which detect, record, signal, and report error within an SoC implementing the Arm RAS System Architecture. It addresses how different system agents, including general-purpose processing elements (PEs), satellite management controllers (SatMCs), and firmware collaborate in the control and recording of RAS events.

4.1 Interrupt routing

Errors detected by RAS-capable components are reported through RAS interrupts, as described in section 2.4, RAS Reporting Mechanisms. Depending on system design and configuration, these interrupts can be routed to different agents:

- Application processor (AP) cores: for in-band handling by either:
  - Firmware (Firmware-First), used when firmware handles the initial error processing
  - Operating System, or Kernel (Kernel-First), used when software directly consumes the interrupt
- Satellite management controllers (SatMC): which can act on behalf of the SoC to perform logging, reporting, or mitigation.

Notably, the interrupt receiver is not always the final handler or the requester that consumes the error. For example, a first application processor (AP) might access memory and trigger detection of an uncorrectable error, which the system defers using poison. The system records the error and issues an FHI or ERI (or both), while the first AP later consumes the poison and raises an SEA or SEI exception. The FHI or ERI may then be handled by a different AP or by the SatMC.

4.2 Error recording

Each RAS node implements one or more standard error records. RAS nodes expose memory-mapped registers (ERR<n>STATUS, ERR<n>MISC<m>, ERR<n>ADDR) that software can use to retrieve the recorded errors.

RAS node registers might also be accessible using system registers in the CPU (ERXSTATUS, ERXMISC<m>, ERXADDR). For example, if the RAS node is implemented as part of the CPU and its caches, or by a component closely coupled to the CPU, such as a DSU.

4.3 System control of error detection and processing

RAS control includes mechanisms for:

- Enabling/disabling error recording.

- Routing interrupts via ERR<n>CTLR
- Managing counters for example Corrected Error Counter overflow triggers.

These capabilities allow software to classify and filter error conditions appropriately. Software can determine the capabilities of a RAS node from ERR<n>FR.

4.4 Error injection

Error injection is a feature in the Arm RAS architecture that enables software to simulate hardware faults for testing system reliability and error handling. It uses registers including ERR<n>PFGCTL and ERR<n>PFGCDN, to inject errors including:

- Correctable single-bit ECC errors
- Uncorrectable multi-bit errors
- Deferred errors

These errors are recorded in standard error records and can optionally trigger interrupts such as Fault Handling Interrupt or Error Recovery Interrupt. Injected errors do not affect actual memory or data flow, they update status registers for safe validation of software error response mechanisms.

4.5 RAS event flow examples

On platforms such as Arm® CSS V3:

- When a corrected memory error occurs in the L1 cache of a CPU, it is logged locally and might trigger a Fault Handling Interrupt (FHI).
- A double-bit ECC error in L2 if deferred, it might trigger an FHI, and if consumed, might trigger an exception, such as a Synchronous External Abort (SEA).
- Errors that are severe enough can result in a Critical Error Interrupt are routed to a system controller.

Each interrupt class can be independently configured for routing and masking, based on system policy.

4.6 System-level reporting

Error reports may be sent to a system administrator node, typically via one of two models:

- In-band reporting: Error status is collected and reported via software running on the AP cores.
- Side-band reporting: Error status is handled and forwarded to the platform's BMC or remote server management interfaces by SatMC.

Arm architecture refers to the management controller as a SatMC. The system utility bus provides access to all RAS records, and BMC or SatMC can be configured to collect data periodically or on event-driven basis. For more detail see Arm Server Base Manageability Requirements 2.1.

4.7 Example integration: Arm® CSS V3

The Arm® CSS V3 platform integrates RAS control across:

- Neoverse CPUs which include RAS extensions, ECC-protected caches.
- CMN-S3AE interconnect supporting SLC and snoop filter error logging.
- GIC-700 supports Fault and Error interrupts.
- MMU-S3 enables fault error tracking.
- MCP/SCP that receive, log, and forward error events.

5. Related information

Here are some resources related to material in this guide:

- Learn the Architecture RAS Overview provides an overview of the RAS Architecture.
- Arm® Reliability, Availability, and Serviceability (RAS) System Architecture for A-profile architecture (IHI0100).
- Arm Architecture Reference Manual for A-profile architecture provides full A-profile architecture details, including RAS.
- Learn the Architecture RAS Software Guide contains information about software side of RAS.
- Arm Server Base Manageability Requirements 2.1 contains more information related to RAS system level reporting.