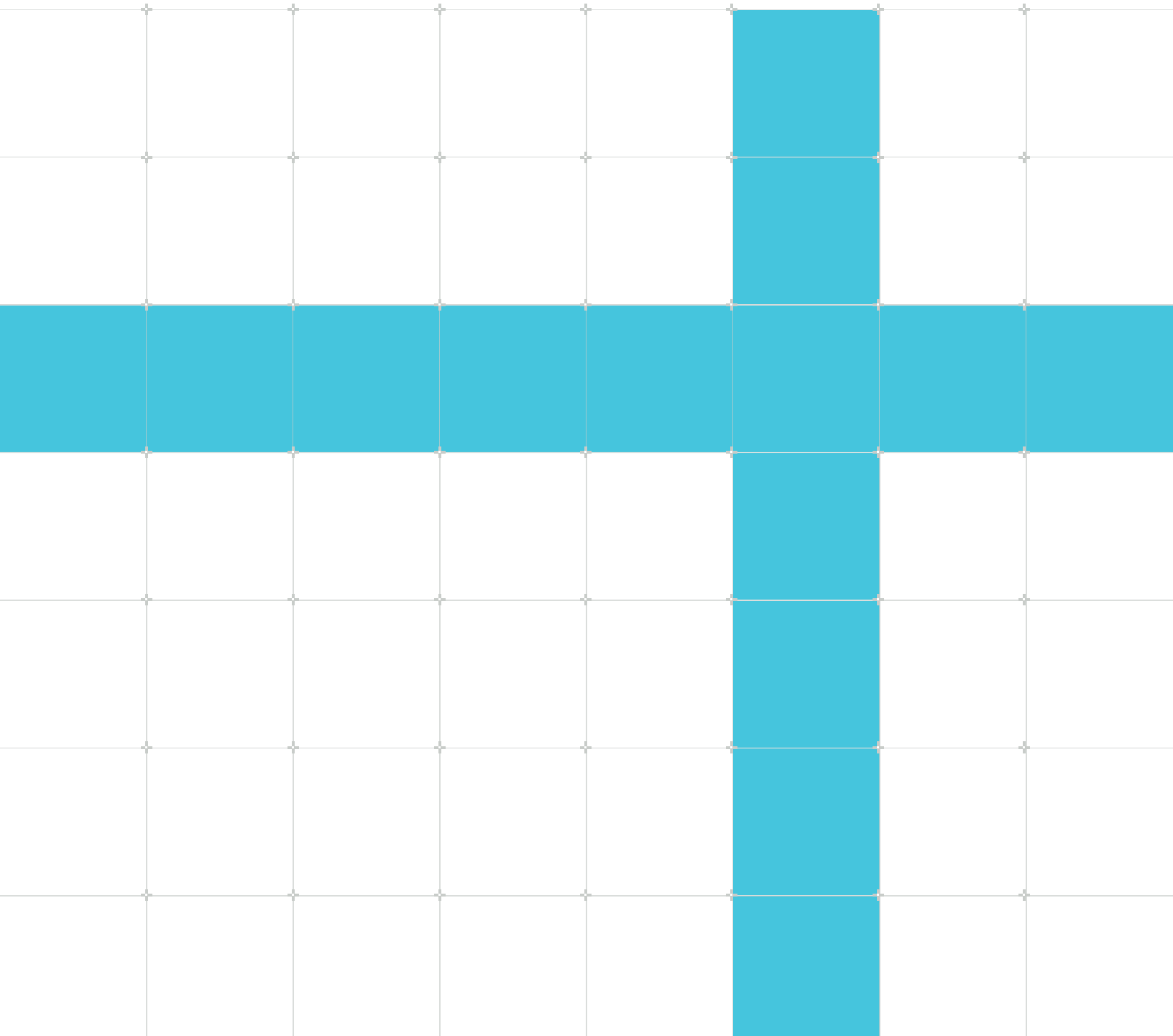
Learn the Architecture - RAS Overview

Version 1.0

Issue 01
107790_0100_01_en



Learn the Architecture - RAS Overview

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	22 June 2023	Non-Confidential	First release.

Proprietary Notice

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 the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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

There are three key attributes of a robust, dependable, computer system: Reliability, Availability, and Serviceability (RAS).

These attributes can be defined as follows:

Reliability

A reliable system consistently provides correct service according to its specification. Results or computations are correct, and arrive within the time allotted to the task.

Availability

An available system is ready to perform its expected function, with as little downtime as possible.

Serviceability

A serviceable system is able to undergo modifications and repair. The system can provide information about its operation to aid in system servicing.

RAS support is essential for many computing situations, reducing unplanned outages as follows:

- Detecting and correcting transient errors before they cause application or system failure. System failure might impact the customer's business and reputation when systems are used for mission-critical functions. In some cases, reliability might form part of the service's value proposition.
- Identifying and replacing failing components. In any system, failures are inevitable. One approach is to replace components on a regular schedule regardless of whether they are failing or not, but this is costly. RAS allows failing components to be identified, and the cost of maintenance can therefore be reduced by only replacing failed parts.
- Predicting failures ahead-of-time to allow replacement during planned maintenance. Scheduled maintenance is much cheaper than unscheduled call-outs.

The RAS extension is a mandatory extension to the Armv8.2-A architecture, and an optional extension to the base Armv8.0-A architecture. Armv9-A and Armv8-R inherit this support for RAS. Armv8.4-A and Armv8.7-A introduce additional architectural features to the RAS extension.

There are two aspects to RAS: the RAS extension to the Processing Element (PE) architecture, and the RAS system architecture.

The RAS extension provides the following architectural features to help systems improve RAS:

- System registers for accessing optional error records and fault injection controls defined by the RAS system architecture
- An Error synchronization event and Error Synchronization Barrier instruction, `ESB`, that software can use to isolate the effects of errors
- A non-maskable asynchronous error exception, for errors reported to the highest Exception level (EL3), and changes to the exception model to ease partitioning of error recovery from other exception handling software

- Additional register fields allow the Processing Element (PE) to report a PE error state when an error exception is taken

The RAS system architecture provides a framework for building RAS features in a system:

- A standard format for error records:
  - Reports the occurrence and severity of errors
  - Provides useful information to software, such as field replaceable unit (FRU) identification.
- An architecture for reporting different severities of error either synchronously as an in-band error response, or asynchronously as error interrupts.
- Standard extensions for error counters, error record timestamps, and fault injection.

The RAS extension lets you design systems with a wide range of RAS capability. For example:

- A system with basic error recovery might provide very few RAS hardware features, and simply reset when an error is detected.
- A mission-critical system which needs to provide high reliability might provide hardware features such as data consistency checks and component redundancy to maximize reliability and availability.

The system designer decides the importance of RAS to their situation, and uses the RAS architecture extension to implement a solution that meets their needs.

2. RAS basic concepts

This section of the guide introduces the fundamental concepts and terminology related to RAS.

2.1 Faults, errors, and failures

When a system behaves as expected, according to its functional specification, then it is providing correct service. Correct service can cover many different aspects of behavior, for example:

- A calculator must produce correct results, showing 2+2=4, rather than 3 or 5.
- A braking system must respond within a specified period of time.
- A banking application must not divulge private information, sending account details on a secure channel.

An error is any deviation from correct behavior. Errors are caused by faults. Faults can be either internal or external to a system, and either transient or persistent. The following table gives examples of each of these fault categories:

Table 2-1: Examples of fault categories

	Transient fault	Persistent fault
Internal fault	On-chip cross talk between bus lines	Manufacturing defect in silicon chip
External fault	Solar radiation causing electromagnetic interference	Malfunctioning power supply

When a fault is activated, it causes an error. Errors that are not detected are called latent errors.

When an error occurs, the error propagates through the system until it results in a failure. A failure is the event of deviation from correct service. This can include data corruption, data loss, and service loss.

The following diagram shows the relationship between faults, errors, and failures:

Figure 2-1: Relationship between faults, errors, and failures

2.2 Error propagation

Errors are propagated in a system when a transaction between a producer and consumer deviates from correct service because of the error.

A simple example of error propagation is when a producer passes a corrupt data value to a consumer.

Other examples of error propagation include:

- The error causes a transaction to occur that should not have occurred.
- The error prevents a transaction from occurring that should have occurred.
- The error causes transactions to occur in a different order.

If the data is corrupted, then an uncorrected error is propagated from the producer to the consumer.

If a system has no error detection capabilities, all errors that occur are latent errors because they are never detected. These latent errors are silently propagated through the system until either of the following happens:

- The error affects the correct behavior of the system, causing a failure. This is a Silent Data Corruption (SDC) failure.
- The error is masked. An error is masked when the behavior of the system is such that the error does not affect the correct service of the system. For example, if an error resulted in an incorrect data value in a register, if that data value was subsequently overwritten by uncorrupted data before the corrupted data is used then the error would be masked.

2.3 Error detection

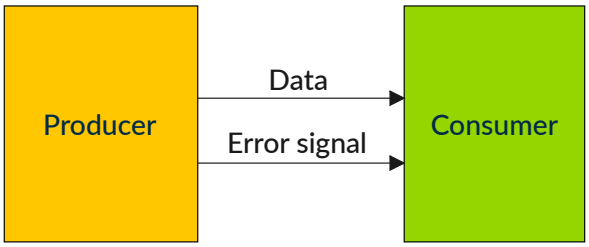
When a component accesses memory or other state, an error might be detected in that memory or state, and corrected, deferred, or signaled to another component as a detected error.

The interface between a producer and consumer might indicate that the data is corrupted, for example, by including an error signal.

If the error signal indicates the presence of the uncorrected error, a detected error is signaled and passed to the consumer. Otherwise, the uncorrected error is silently propagated.

The following diagram shows data being passed from a producer to a consumer together with an error signal.

Figure 2-2: Error propagation from producer to consumer, with error signal



Note

The error signal might be a separate signal, or embedded in the data as an error detection code.

2.4 Infected and poisoned

When a consumer receives an uncorrected error propagated from another component it consumes the error.

If an 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 signals a detected error, the state is poisoned.

2.5 Containable and uncontainable

If an error has been silently propagated, we say it is uncontained.

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

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

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

2.6 Error classification

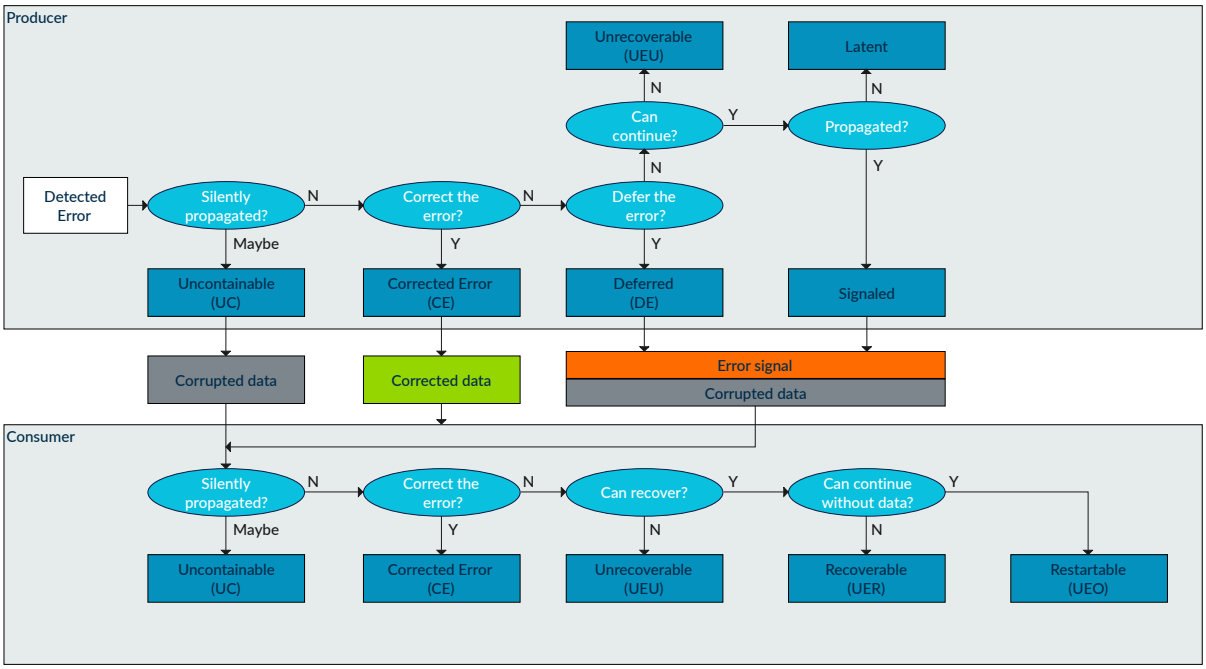
The RAS extension defines the following error states:

- Uncontainable (UC)
- Unrecoverable (UEU)
- Recoverable (UER)
- Restartable (UEO)
- Deferred (DE)
- Corrected (CE)

When an error occurs, it always starts with a producer. If the producer is using error detection techniques, for example a parity bit on data, then the producer might be able to detect the error. In this case, the error is detected. The error is classified based on how the error is dealt with by both the producer and the consumer.

The following flow chart shows the categorization process:

Figure 2-3: Error categorization



The categorization process starts with the producer. Depending on the nature of the error and the capabilities of the system, the error can be categorized as follows:

1. Might the producer have silently propagated the error?

If the error might have been silently propagated, the error is uncontainable (UC). For example, if corrupted data has been received by the consumer, or corrupt data might be consumed by any other agent, without the consumer knowing that the data is corrupted.

2. Can the producer correct the error?

   If the error can be detected, the producer might be able to correct the error, for example by using a cyclic redundancy check (CRC). In this case, the error is a corrected error. The corrected data is passed to the consumer, and normal operation continues. In this case the error is a corrected error (CE).

   Otherwise, if the error cannot be corrected, it is an uncorrected error.

3. Can the producer defer the error?

   An error is deferred by hardware if hardware can make forward progress without consuming the error.

   Deferred errors result in corrupted data being passed to the consumer, but with an error signal to alert the consumer. The consumer can then decide how to deal with the error. For example, if it does not need to consume the corrupted data, it can leave the error as deferred.

4. Can the producer recover from the error?

   If the error prevents the producer from continuing operation, then the error is categorized as a detected unrecoverable (UER) error.

5. Does the producer propagate the error to a consumer?

   If the detected, uncorrected, and undeferred error is not passed to a consumer, then the error is a detected latent error.

   Otherwise, if the error is passed to the consumer, it is a signaled error. Corrupted data is passed to the consumer, but with an error signal to alert the consumer.

If the consumer receives corrupted data, then the error is further categorized based on how the consumer deals with the error:

1. Was the error silently propagated?

   If the error might have been silently propagated, the error is uncontainable (UC).

2. Can the consumer correct the error?

   If so, the error is a corrected error (CE).

3. Can the consumer recover from the error?

   If the error has corrupted the component's state and recovery of the component is not possible, the error is an unrecoverable error (UEU).

4. Does the consumer need the corrupted data to continue?

If the consumer can take action to recover from the error, but requires the corrupted data to make progress, the error is a recoverable error (UER).

If the consumer does not require the corrupted data to make progress, and therefore can recover with no action, the error is a restartable error (UER).

3. Arm RAS extension and system architecture

RAS provides a framework that lets you design and build RAS features in a Processing Element (PE) implementation and the system around it.

There are two aspects to RAS: the RAS extension to the Processing Element (PE) architecture, and the RAS system architecture.

The RAS extension does not define the level of reliability, availability, and serviceability in a PE. The RAS framework allows for a wide range of RAS capabilities, from basic RAS approaches such as “reset-on-error” through to very high-reliability systems with sophisticated error handling and correction features. But the specific RAS features that the PE includes are **IMPLEMENTATION DEFINED** and must be decided by each individual designer. The RAS extension provides the framework for the implementation to communicate these choices to software when an error occurs.

At a high-level, the RAS extension provides the following:

- Exception reporting mechanisms for fault handling and error recovery
- ISA extensions to enable the isolation of uncontainable errors
- Standardized error records for use by error recovery and fault handling
- Fault injection, from Armv8.4-A

Each of these is discussed in the following sections.

3.1 Exception reporting

When an error is recorded outside of a PE, one or more of the following might be generated and sent to the PE:

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

The error can also be deferred by generating a poison value. Data poisoning is a mechanism for marking data as corrupted. The poison value is stored with the corrupted data to indicate that an error was detected. Subsequent accesses of the data see the poison value and treat it as a detected error.

3.2 ISA extensions

The RAS architecture extension introduces the Error synchronization event and the `ESB` instruction.

An Error synchronization event ensures that all SError Interrupts that would be reported as unrecoverable (UEU) are either taken or pended before execution continues.

The `ESB` (Error Synchronization Barrier) instruction generates an Error synchronization event to isolate UEU errors. This ensures that interrupts which are generated by instructions before the `ESB` are either taken or pended before execution continues beyond the `ESB` instruction.

Implicit ESB events are inserted at exception handler entry and exit. This enables errors to be isolated without having to modify software to add explicit `ESB` instructions.

3.3 Error recording

Errors are recorded at the point they are detected, even if they are deferred or corrected.

The RAS extension provides a standardized error record format for recording errors. The information logged in these records includes the following:

- An error code to identify the specific error that was detected.
- Any memory address associated with the error.
- An optional timestamp when the error was detected.
- An optional counter, for counting correctable errors so they can be serviced less frequently.
- Miscellaneous fields used to store application-specific information about errors. These are **IMPLEMENTATION DEFINED**.

These standardized error records allow for:

- Logging, diagnosis, and categorization of faults by severity
- Identification of Field Replaceable Units (FRUs)

The exact number of error record registers is **IMPLEMENTATION DEFINED**.

3.4 Fault injection

To test fault-handling mechanisms, you might want to artificially introduce faults or errors into a system. Fault injection is the deliberate injection of faults into a system for testing. This lets you check that the fault is detected, reported, and then handled correctly.

4. RAS error reporting flow

At a high level, the process of reporting an error within the RAS framework is as follows:

1. An error is detected at a component and reported to a node.

   Examples of nodes include the following:

   - Memory controllers
   - System caches
   - CPU caches
2. The node stores information about the error in an error record.

   This information includes the following:

   - Severity
   - Address
   - Field Replaceable Unit (FRU) data
   - Optionally, a Corrected Error (CE) counter
3. If possible, the node corrects the error. If the error cannot be corrected, then if possible the node defers the error by generating poison.
4. The node raises a Fault Handling Interrupt (FHI) to report that it has recorded the error in the error record.

   If a CE counter is implemented, then an FHI is generated only when the CE counter overflows.
5. If the error cannot be corrected or deferred, then the node does one or both of the following:

   - Returns an in-band error response to the access
   - Generates an Error Recovery Interrupt (ERI)
6. When a PE receives an in-band error response, it generates either a Synchronous External Data Abort (SEA) or an asynchronous SError Interrupt (SEI), depending on the implementation of the PE.

Software can control some aspects of this flow, for example disabling generation of one or more of the interrupts.

5. Example RAS implementations

The RAS extension enables the design of systems with a wide range of RAS capabilities.

This section of the guide describes examples of systems with different levels of RAS capability:

- Fundamental
- Advanced
- Safety-critical

The following table describes these different example systems and the types of RAS hardware features they might implement.

Table 5-1: Example RAS implementations

RAS capability	Fundamental	Advanced	Safety-critical
Areas of deployment	End-user devices. Mobile clients.	Wireless networking infrastructure (5G/6G). Enterprise networking. Data centers. HPC systems.	Safety-critical automotive. Aerospace. High-end industrial automation.
Example systems	Mobile phones. Large-screen compute. Consumer routers.	5G tower. Server blade in datacenter or HPC. DPUs.	Mission-critical systems. Autonomous driving.
Hardware characteristics	Battery powered. Small devices. Low-core counts.	Medium to very high core counts. Deployments from on top of cell towers to secure, cooled data centers.	High-compute performance in small form factors. Redundant hardware environments.
Software characteristics	Some vertical integration of the software stack is possible.	From vertically-integrated software stack to serverless, fully disaggregated, VMs.	Tightly controlled, verified, vertically-integrated software stack.
Active runtime expectation (average)	~10h per day.	24/7 runtime, short of maintenance.	Ranges from ~8h to 24/7 runtime, short of maintenance.
Lifetime expectation (average)	~2y lifetime.	~5-10y lifetime.	~15-20y lifetime or longer.

RAS capability	Fundamental	Advanced	Safety-critical
Serviceability characteristics	Device or part of device replacement.	Will require likely part replacement for a fully-functional component. Serviced by professional and trusted personnel with a high degree of automation. Transient errors, for example errors that do not recur after reset, might not require component replacement. Diagnosis and recovery procedures handled remotely with high level of automation support. Regular scheduled maintenance and replacement is a common practice.	Expected to be serviced by qualified and trusted personnel. Needs protection against unauthorized access to parts. Parts replacement is common practice.
Usage models	Individual devices. Part-time usage.	Varying scale deployments, including very large-scale. Hyperscalers operate 24/7 but could tolerate some faulty parts being taken offline. RAS has high impact on total cost of ownership. Service Level Agreements (SLAs) with customers might require real-time RAS features to meet the agreed service standards, for example not dropping video streams or keeping a cell tower operational 24/7.	Used mostly in drive mode, interspersed with long off time. Reliability and safety are critical while being driven. Long term reliability important because of high cost to owner.
Risk of failure	User annoyance. Productivity loss. Reputational loss.	Impact to reputation. Potential liability. Potential high cost due to service availability (for example, search engines) or having to re-run large-scale jobs (for example, weather modelling). High financial impact due to part cost and cost of replacement.	Potential liability and reputational loss. High cost to recall affected parts, if needed.
Approach for error and fault handling	Restart the device. If a fault is permanent, or in a critical component, then device replacement is necessary.	Isolate the faulty component, or the faulty location within the component. For example, with errors in memory, the operating system or hypervisor might place the page containing the faulty location into a quarantine list and does not allow software to use the location again. It might even recover the data that was in the page, for example from a disk copy. Attempt to contain errors. If the level of errors becomes significant, migrate all other running threads off the device. If a fault is permanent, if desired, operation without the component must proceed.	Usually requires restarting the device. Faults and errors must be detected rapidly. The faulty device must quickly and safely get into a safe state. A safe state might mean not losing control of the system, that is a fail safe mode of operation.

RAS capability	Fundamental	Advanced	Safety-critical
Types of RAS features	SW protections. Poison/Deferred errors. Error reporting. RAS architecture.	All fundamental features. End-to-end error protection. Symbol-based ECC. Extra DRAM protections. Error logging and out-of-band export.	All advanced features. Safety features, including dual-core lockstep and other safety features.

6. Related information

Here are some resources related to material in this guide:

- Reliability, Availability, and Serviceability (RAS) Arm Architecture Reference Manual Supplement