NVIDIA Grace RAS Overview

Application Note

Document History

DA-10735-001_08

Version	Date	Authors	Description of Change
01	December 2021	VS, RK	Initial Release
02	March 2023	VS, RK	Added ACPI Error Injection details
03	January 2024	NB, RK	• Added NVIDIA® NVLink®. • Revised the ACPI Error Injection section. • Updated the Firmware First RAS Flow section.
04	February 2024	AK, MP	• Added EINJ usage examples. • Added UEFI MM CPER Error routing information.
05	April 2024	AK, NS	• Added channel sparing policy and thresholds. • Added HSM Error Injection. • Added DCC-ECC Error injection. • Added CPER Decoding section.
06	July 2024	NS	• Added the following sections: > Reporting Mechanisms. > SOCHUB Error Injection. > Controlling HSS using Error Injection. > Controlling PCIe Leaky Buckets using Error Injection. > DPC mechanism for uncorrectable PCIE errors.
07	February 2025	NS	• Updated PCIe EINJ section. • Update decoded CPER examples. • Added the following section: > Throttling of PCIe Correctable Errors. > Mapping for the UEFI Memory CPER fields. > DRAM error reporting using CPERs.
08	August 2025	NS	• Added channel sparing policy and thresholds. • Added a table to specify possible ACPI CPER severities for different types of CPERs. • Minor fixes and updates to the existing text.

Table of Contents

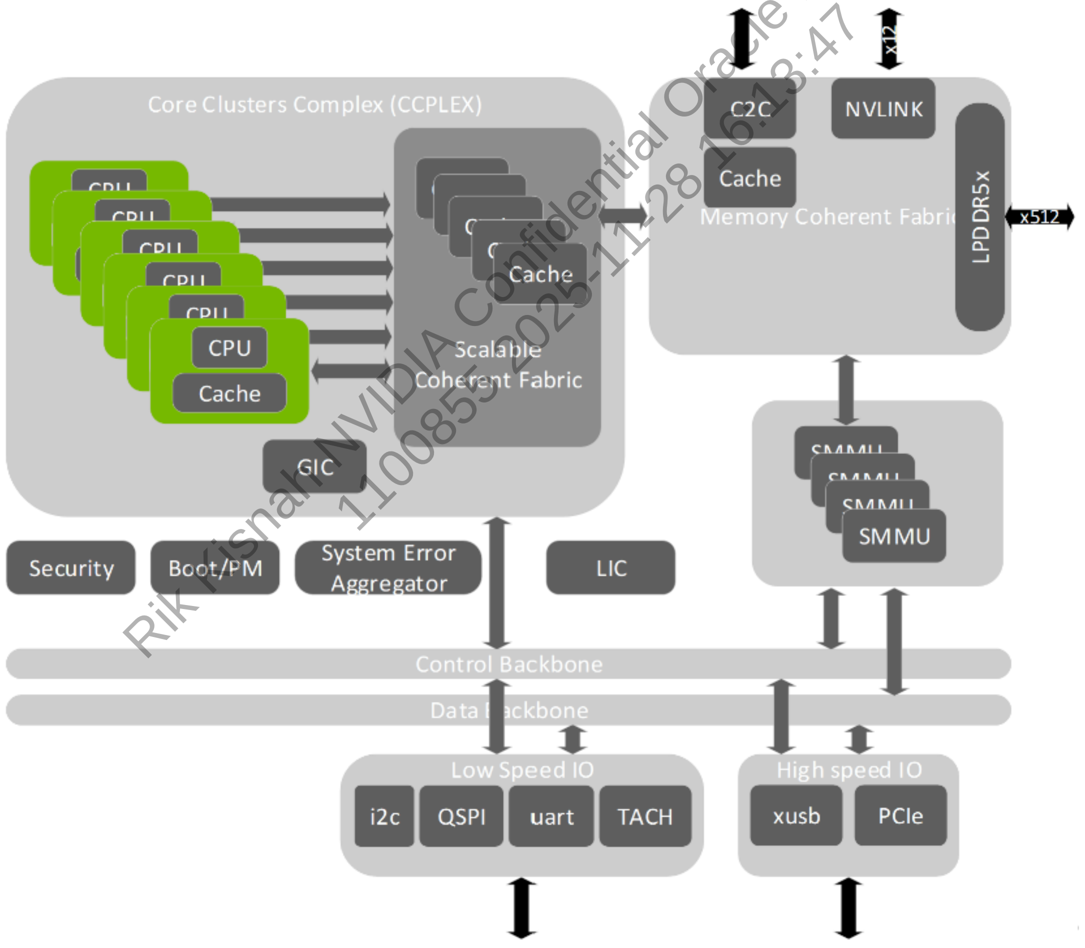
List of Figures

List of Tables

Grace Overview

NVIDIA® Grace™ is a Server Class Arm® Compute System-on-Chip (SoC) that is designed to operate as a companion to the NVIDIA Hopper graphics processing unit (GPU). It also can operate as a stand-alone server SoC that is targeted for the Cloud Server market. Figure 1 shows a high-level diagram of the chip.

Figure 1. Grace Block Diagram



This release includes the following features:

> Up to 72 Arm v8 class cores

- Up to 100+ MB L3 cache
- Up to 496b of LPDDR5x (30 channels)
- Support for up to 4 coherent sockets using NVIDIA® NVLink®
- Up to 68 lanes of PCIe Gen5
- 10 lanes of C2C support for NVIDIA GPU
- GIC700
- Arm Reliability, Availability, and Serviceability (RAS) v8.4 support

About Grace RAS

RAS Terminology

Table 1 defines the Arm RAS terminology that will be used in this application note.

Table 1. Arm RAS Terminology

Term	Definition
Correct Service	Delivered when the service implements the system function.
Failure	The event of deviation from correct service (data corruption, data loss, and service loss).
Error	The deviation from the correct service.
Fault	The cause of the error.
Transient Fault	A fault that occurs at a moment in time.
Non-transient/Persistent fault	A fault that occurs all the time.
Correctable Error	An error that is corrected by the hardware.
Deferred Error	Errors that are detected and marked as poisoned, so that future consumption of these errors can be handled.
Uncorrectable Error	Errors that cannot be recovered from by hardware or software mechanisms.
Error Propagation	The act of transferring an error from a producer to a consumer.
Infected	Data that has incurred a detectable error and is no longer in a correct state.
Poison	Marking infected data so that the additional consumption of this data will be detected as an error by the consumer.
ERI	Error Recovery Interrupt.
SEA	Synchronous External Abort.
SEI	System Error Interrupt.

CPU: Arm RAS v8.4 Compliance

The *Arm® Architecture Reference Manual Supplement - Reliability, Availability, and Serviceability (RAS), for Armv8-A* describes the correct service-as-a-service that might include producing correct results, producing results in the time allotted to the task, and not divulging secrets or secure information.

The purpose of the Grace SoC is to adhere to this compliance for all on-die OS visible Arm Processing Elements (PEs) and across the system cache hierarchy. This includes adherence to the Arm RAS Standard Error log across our Core Compute Complex and across our PCIe Root Ports. In addition to the Arm RAS records, Grace supports many internal error logging capabilities that are logged by using an internal proprietary format that is shared for safety purposes.

Internal Cache Protection

Grace contains multiple levels of cache hierarchy (L0, L1, L2, L3, and so on). As a rule, Grace supports Single-bit Error Correction (SEC) and Double-bit Error Detection (DED) across all data caches (data and tag).

Here is a list of the features:

> L1 Instruction Cache Data: SEC and invalidation/refetch

> L1 Instruction Cache Tag: Parity

> L1 Data Cache Data/Tag: SEC/DED

> L2 Data Cache Data/Tag: SEC/DED

> L3 Data Cache Data/Tag: SEC/DED

> Compliance to Arm RAS Standard Error Format

Single-bit errors are corrected and logged, and double-bit errors on the data results in Data Poisoning of the line. The poison indication remains with the data as it moves through the system.

DED errors on Tag caches result in slightly different behavior based on the hierarchy of cache. For L1/L2 Tag double-bit errors, the address and data are lost, so the line in the cache is invalidated, which causes an Error Recovery interrupt. If the double-bit error occurs at the L3, Grace will enter containment mode, which prevents additional error propagation. The slice of L3 where the error occurs will stop responding to transactions.

You can recover from both error types, but a system reset will be required at some point. Based on the error location, some level of logging and firmware might be possible before a system reset.

DRAM Features

The memory technology on Grace is LPDDR5x. This technology is different from the technology that is used on a Server Class SoC, so there is a slightly different approach to RAS.

Dynamic Random-Access Memory (DRAMs) are now soldered down on the same module as Grace, so a reduction in the errors that typically occur at the socket and channel level is expected. Studies imply that socket and channel errors are the largest contributors to the number of errors (Meza, Wu, Kumar, & Mutlu, 2015). Features such as error correction code (ECC), scrubbing, poisoning, and page off lining techniques have also been incorporated.

Here is a list of the features:

> Form Factor: To improve performance and minimize socket and channel errors, the DRAM devices have been soldered down.
> ECC: SEC/DED ECC algorithm to reduce multi-bit aliasing and correct errors.
> Data protection: Demand and Patrol Scrubbing capabilities.
> Poison support.
> Aggressive page offlining is completed by using the firmware/kernel.
> Multiple single-bit or one double-bit errors on a page will cause the page to be removed from the memory map.
> Error counting and logging features.
> Internal bus protection using Parity/ECC.
> Internal RAM protection using ECC.
> Internal register protection using Parity.

Channel Retirement and Sparing

Channel sparing allows you to reserve some memory channels as spares. When a memory channel exceeds the error threshold, it is retired, and a spare is activated to replace the retired channel. This section provides information about the channel sparing policy.

In the Grace SBIOS, to enable or disable channel sparing, click **Device Manager > NVIDIA Configuration > Grace Configuration > Disable Channel Sparing**.

Caution: On certain platforms, when you disable sparing channels, the memory channel might be permanently retired instead of being spared.

For the SKU A02P chip, based on the number of available channels, the system will spare some channels to arrive at 30 channels with which to boot.

For the SKU A01/A02 chips, channel sparing is always disabled, the system maintains no spares and always uses all available channels.

Regardless of the number of channels that are set as spares, the system always ensures that 30 memory channels are available to boot.

Channel Limit

Up to four Page Retirements are allowed for each GB. After this threshold is exceeded, the channel will be permanently retired.

System Limit

Up to two Page Retirements for each GB are allowed across all channels in the entire system. After this threshold is exceeded, the system might be evaluated for RMA.

Page retirements are calculated on 64 KB pages using the following formula:

```
Max Memory Lost = 128 KB / 1 GB = 0.012%
```

Table 2 provides more information for a 30-channel system.

Table 2. Page Retirement for a 30-Channel System

GB*/Channel	Channel Retirement Limit (# of Page Retirements)	Total Capacity	Max Retirements	Max Memory Lost
4	16	120 GB	240	15 MB
8	32	240 GB	480	30 MB
16	64	480 GB	960	60 MB

PCIe RAS Features

Grace supports up to 68 lanes of PCI Express (PCIe) Gen5 and supports all standard PCIe Gen5 features, with additional optional support and many internal features.

Here is a list of the features:

> All standard PCIe Gen5 Error mechanisms.

> PCIe Advanced Error Reporting.

> Downstream Port Containment.

- An Arm RAS extension with Error logging as defined by (ARM, SBSA ECR: Error handling requirements for Root ports, host bridges, Root Complex Integrated Endpoints and Integrated Endpoints behind a root port, 2018).

  Refer to the *Arm Reliability, Availability, and Serviceability (RAS) Specification Armv8,* for the Armv8-A architecture profile on https://developer.arm.com/documentation/ddi0587/latest for more information.
- Internal Bus protection using Parity and ECC.
- Internal RAM protection using Parity and ECC.
- Timeout Checks.
- Internal bus protocol checking.
- Poison handling.
- Additional internal checks.

As implied, logging follows the standard PCIe formats when PCIe defined failures occur **or** by using the Arm RAS extensions format when errors in the Root Port or host bridge occur.

Coherent Link Features: NVLink and C2C

Grace communicates between multiple Grace sockets and has a custom connection between Grace and Hopper (C2C). Protection is maintained across the link and in the internal data structures.

Here is a list of the features:

- Link Training error detection (NVLink).
- ECRC Data link checking.
- Transaction level protocol checking (NVLink).
- Poison forwarding.
- Programmable options to enter the containment mode for a variety of errors.

  In the containment mode, uncorrectable errors can lock down the link and prevent errors from moving beyond the socket.
- The Hopper Containment mode (C2C only) allows for an isolated reset.
- Internal memory protections by using parity and ECC.
- Internal bus protections by using parity and ECC.
- Timeout errors.

Logging errors in NVLink occurs by using a custom format that is consistent between all links in Grace and Hopper. Errors are routed to the SoC safety aggregator or directly to the Legacy Interrupt Controller.

Internal System Bus Integrity

Grace implements extensive protection along command and data paths in the following ways:

> Most Control and Data paths in the design are protected with ECC, Parity, or a mixture of both.

> Data structures (FIFOs and RAMs) are also generally protected by using ECC or Parity.

> Protocol and other checks are implemented on multiple internal fabrics to identify errors that might occur in the system.

Here is a list of the features:

- Parity/ECC across internal RAMs
  - Minimum of Parity on Control
  - ECC on Data
- Protocol checking
- Lockstep checking across busses that support Lockstep devices
- Timeout checks
- In-band error reporting

Across the design, this constitutes thousands of error protection and detection sources that are coalesced through a System Error Aggregator and that feed into the Legacy interrupt controller. Errors that occur in this region are logged by using the NVIDIA Error Collator format.

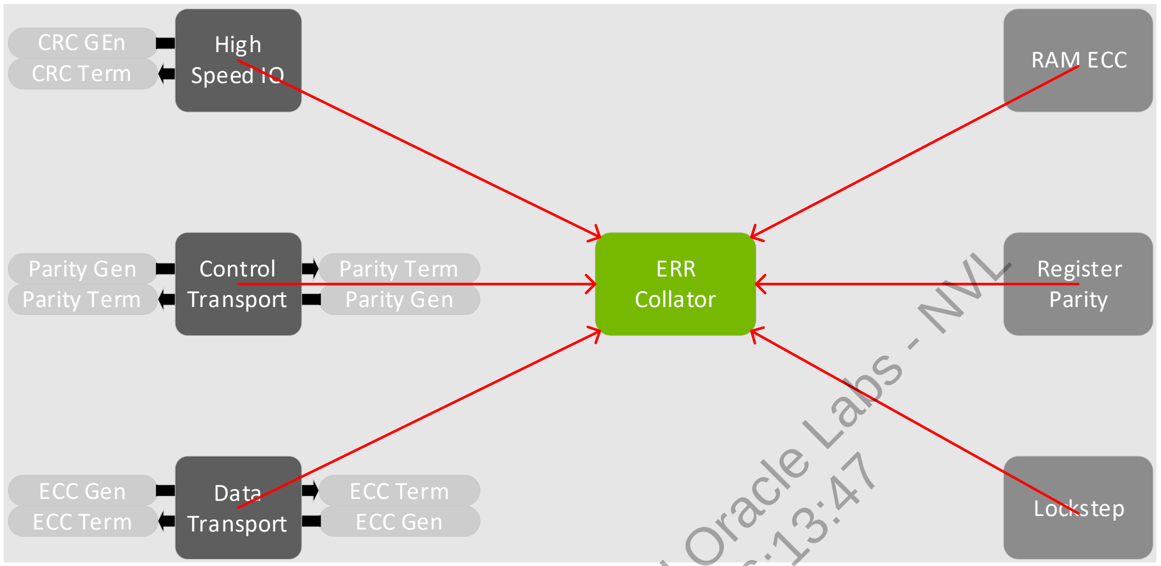
SoC Error Aggregator Features

In Grace there are thousands of error checks that need to be aggregated across the design and can be classified into the following groups:

- Consistency Monitors
- Cyclic redundance checks (CRC)
- Thermal Events
- Watchdog Timers
- Access Protections
- Lockstep Monitors
- ECC/Parity Checks
- Clock/Voltage monitors
- Timeout errors
- Other miscellaneous errors

A standard mechanism is used by many units in the design to implement the previously mentioned checks, and this standard mechanism has a local error collator function and many optional plugins (refer to Figure 2). Each source feeds into a local error collator that creates a standard logging mechanism.

Figure 2. Local Error Collator and Plug-Ins



Errors are passed upstream to the System Error Aggregator, which aggregates errors from around the SoC and notifies the firmware/software by using the Legacy Interrupt Controller as shown in Figure 3. The errors are propagated to the generic interrupt controllers (GICs) as shared peripheral interrupts that can then be handled by reading a common set of registers.

Internal Memory Protection

With the aim of protecting information, most of the internal memory in Grace is protected with parity or ECC. In the main core and cache domains, this information is fed into the Arm RAS records and routed to the GIC as an interrupt. The errors are fed into the NVIDIA System Error Aggregator, which is logged in an NVIDIA proprietary format and is sent to the Legacy Interrupt Controller.

Internal Register Protection

Many of the internal registers in the SoC are protected with Parity, which prevents transient errors that might cause invalid operations in the design. Register parity protection is part of the standard NVIDIA Error Collator plugin that is used across the design.

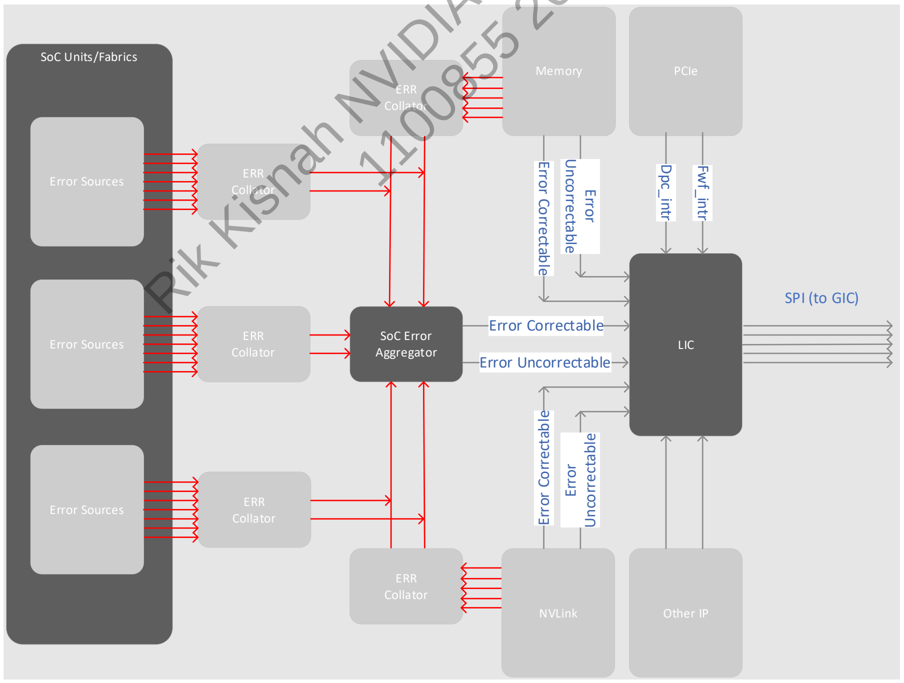
Error Reporting

Interrupt Routing

In Grace, there are multiple levels of error collection and interrupt routing. Many SoC IP errors are routed to the SoC Error Aggregator before asserting an interrupt to the Legacy Interrupt Control (LIC) which turns into a Shared Peripheral Interrupt (SPI). PCIe, Memory, NVLink, or another error that reside in the SoC, but not in the Core Compute Complex, interrupt enabling, control, and logging is primarily completed in the source Unit.

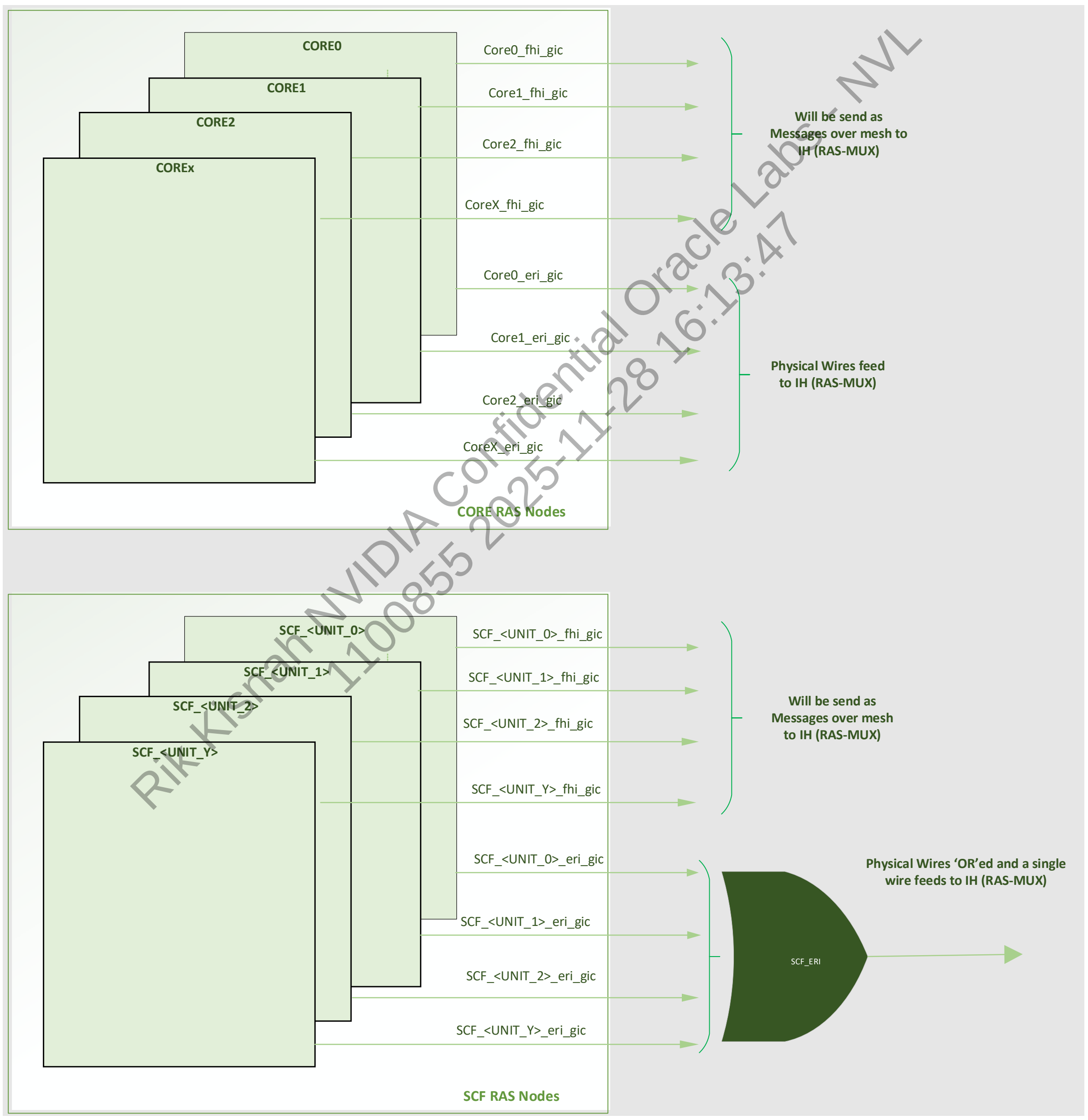
These errors are routed to the legacy interrupt controller, which aggregates SoC interrupts and are converted to SPI. Figure 3 shows an example of how SoC interrupts can be steered in Grace. These SPIs are converted to ERI or FHI later, based on uncorrectable or correctable error types.

Figure 3. SoC Error Interrupt Routing



For errors that occur in the Core Compute Complex (CPUs, Caches, Coherent Fabric, and support logic), the errors are routed to a RAS mux and sent to the GIC as Private Peripheral Interrupt (PPIs). Figure 4 shows the CPU and other RAS interrupt sources in the Core Compute domain. Before entering the PPI Redistributor, errors can be treated as Fault Handling Interrupts (FHIs) or Error Recovery Interrupts (ERIs). The plan is to route the system-level, uncorrectable errors to error recovery interrupts and route correctable errors to Fault Handling Interrupts.

Figure 4. RAS Interrupt Sources Inside the Core Compute Domain



Reporting Mechanisms

This section provides information about reporting mechanisms.

Rate Limiting Mechanism for Identical CPERs

Identical CPERs that occur in the system follow a rate limiting mechanism.

When the firmware generates a CPER, its content is compared to the previous CPER. One of the following processes will occur based on the results of the comparison:

> If the contents of the current and previous CPER are identical (timestamp is excluded in the comparison), the timestamp is compared.
  - If the time elapsed between the two identical CPERs is less than 1 second, and the severity of the CPER is fatal, firmware assumes that the OS is unable to restart the system, and a firmware initiated L2 reset is triggered.

    If the time between two identical CPERs is less than one second, and the severity of the CPER is not fatal, the occurrence count of the CPER is incremented.
  - If the occurrence count is under the reporting threshold (initialized to 5), the new CPER is dropped.
  - If the occurrence count reaches the threshold, the CPER is reported, and the threshold is doubled.
  - If the time elapsed between the two identical CPERs is greater than 1 second, the threshold is re-initialized to 5, and the occurrence counter is set back to 0.

> If the CPER contents are different, the occurrence counter and thresholds are reset to initial values.

Leaky Bucket Mechanism for PCIe Correctable Errors

PCI Express correctable errors use a leaky-bucket mechanism with a counter size of 32 and a drain rate of 1 error per minute. If the counter overflows, one error is reported, and the bucket is reset. Each PCIe controller will have its own leaky bucket.

When injecting PCI Express correctable errors, an error might be reported before all 32 errors are injected. This behavior indicates that the system might be observing real errors.

Note Error reporting will always follow the leaky bucket mechanism first for PCIe Correctable Errors. After the bucket overflows, if the CPERs are identical, error reporting might be suppressed by the rate limiting mechanism.

Throttling of PCIe Correctable Errors

RAS firmware throttles the reporting of PCIe correctable errors, which are tracked per device. In a four-minute interval (240 seconds), the bit errors are sampled in the following order:

1. 6 seconds, error limit: 1 error
2. 20 seconds, error limit: 5 errors
3. 40 seconds, error limit: 13 errors
4. 60 seconds, error limit: 22 errors
5. 120 seconds, error limit: 39 errors
6. 180 seconds, error limit: 76 errors
7. 240 seconds, error limit: 98 errors

In a sample interval, when the total number of detected errors exceed the error limit for that interval, an internal over-limit counter will increment by one. When over-the-limit counter reaches 5, the PCIe device CE reporting is disabled. The counter for the PCIe device is reset to support the device hot plug. If after 240 seconds, the over-the-limit counter does not reach 5, the process will repeat but over-the-limit counter will reduce by one.

Reporting PCIe-ANF Correctable Errors

The Advisory Non-Fatal (PCIe-ANF) errors are a non-fatal error type that serve as an advisory notification to software. These errors are handled separately from PCIe link correctable errors and will not be a part of the PCIe CE leaky bucket or throttling mechanisms mentioned in previous sections.

PCIe-ANF error reporting follows a separate rate limiting mechanism. When an error is reported by a device, it is checked against a threshold (currently set to 3). If the threshold is exceeded, the ANF error will not be reported, and this is implemented by masking the AER ANF correctable error enable bit for the device. This mechanism is implemented on a per-device basis.

Error Logging

Grace logs errors in multiple formats. Arm RAS error logs are maintained for the Arm PEs, caches, and the PCIe. The rest of the system enables and logs errors (refer to Table 3 for more information).

Table 3. Error Recording Format

Error Logging Location	Error Recording Format
CPU (including L0/L1/L2)	RAS Error Record
CPU Coherent Fabric (including L3)	RAS Error Record
Boot/PM processor subsystem	NVIDIA Error Collator Format
Clocking	NVIDIA Error Collator Format
Control Backbone	NVIDIA Error Collator Format
Data Backbone	NVIDIA Error Collator Format
Memory Controller	IP Specific Registers
Memory Coherent Fabric	IP Specific Registers and NVIDIA Error Collator Format
NVLink	IP Specific Registers and NVIDIA Error Collator Format
Chip2Chip	IP Specific Registers
Security	Nvidia Error Collator Format
PCIe	RAS Error Record and PCIe Architecture
Low Speed Peripherals	NVIDIA Error Collator Format

Error Handling Details

The general policy for error handling in Grace is that errors are logged at the detection source. These errors can enable various interrupts (ERI or FHI), and in many cases are also handled synchronously by the cores. When possible, errors are deferred so that they can be handled when they are consumed by a processing element.

Synchronous Versus Deferred Errors

Where possible, errors are handled in-band, which means that when the cycle completes, you will receive a notification that a PE will act immediately. Examples of this process includes reads that incur double-bit errors in a data cache. Here, data is returned with a poison indication, which causes a Synchronous External Abort (SEA).

When an error occurs on a store type of instruction, the data is marked as poisoned, is deferred until consumption, and no action is taken. (Although, if enabled, there is an option to cause an Error Recovery Interrupt). After a PE consumes that location, an SEA will occur.

Software Flows for RAS Error Handling

Using Firmware First for a Self-Hosted Hopper Instance

Grace does not implement standardized Armv8 RAS-compliant error registers outside the CPU Complex, so you cannot rely on the kernel-first RAS. Instead, you need to rely on firmware to detect errors and abstract them for the kernel. This approach is called Firmware First RAS.

The hardware also does not sanitize Secure World-caused RAS errors, which results in the spilling of Secure world addresses (side channels). Firmware First RAS does sanitize the Secure World errors from the abstracted view that is sent to the hypervisor but still allows for reporting to a BMC.

Grace CPU Secure Firmware acts as the Firmware First RAS firmware, and the Arm Trusted Firmware manages the RAS interrupts and RAS CPU exceptions. This information will be forwarded to a RAS Firmware Secure Partition that is running on the CPU core in Secure EL0/1.

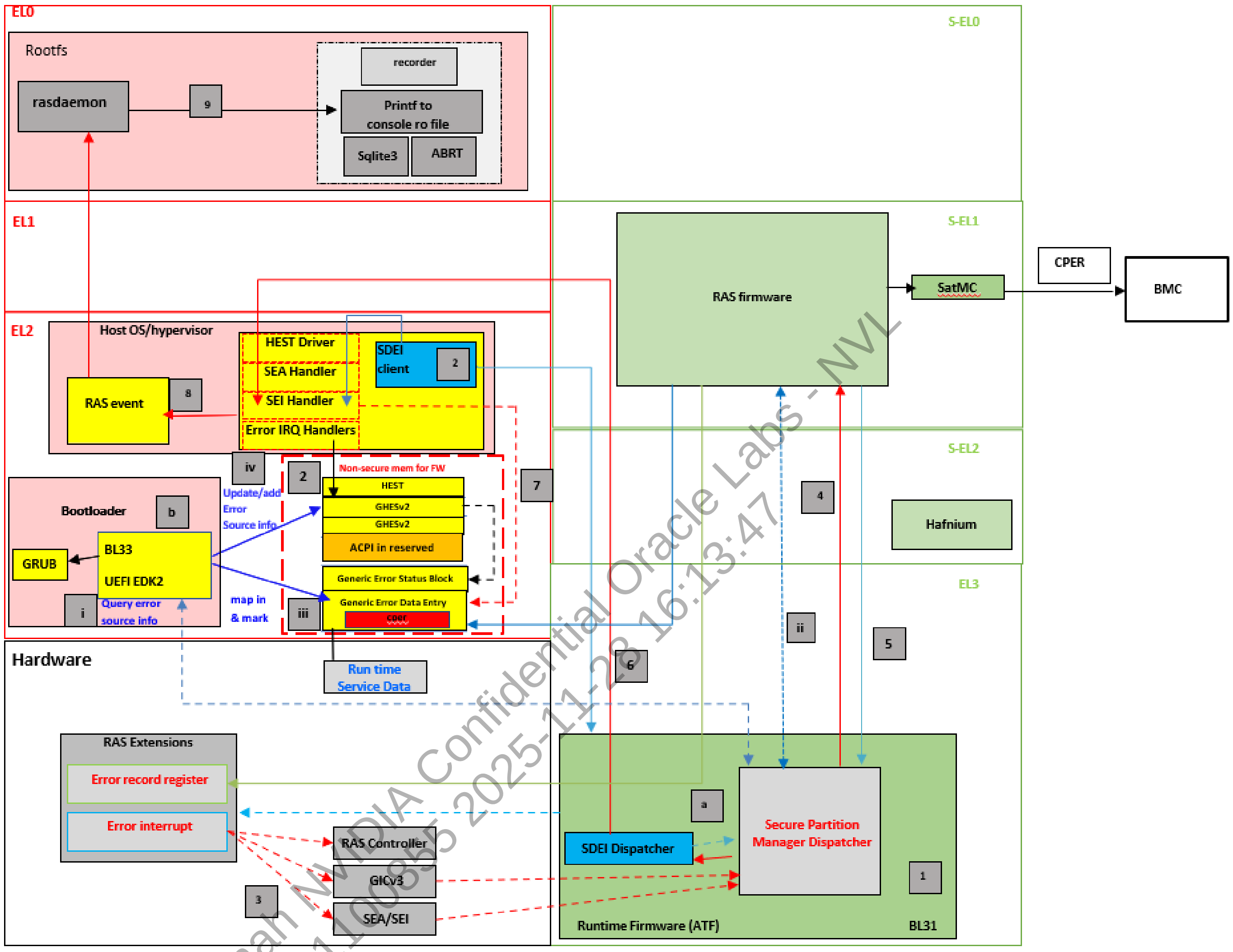
Arm Firmware First Flow for RAS Errors

Figure 5 shows the Arm CPU's execution levels. An unsecure Hypervisor and the Linux kernel are running in the red boxes on the left, and Secure world is running in the green boxes on the right.

Note: On Demeter with VHE, the Host Linux kernel will be in NS EL2.

RAS-related external interrupts and CPU exceptions are routed to the secure EL3 and then to the secure EL0 RAS firmware for error processing. Common Platform Error Record (CPER) logs are generated to be presented to the kernel at an address that is known to the kernel from ACPI GHESv2 table.

Figure 5. Arm v8 CPU Execution Levels



Here are the steps for the first flow in Figure 5:

1. The system boots.
   a. BL31 initializes the Secure Partition Manager dispatcher and the Software Delegated Exception Interface (SDEI) dispatcher.
   b. The following steps occur in the UEFI (BL33), DXE, and UEFI Platform Driver:
      i. The RAS Firmware Secure Partition Image is queried for error source information.
      ii. The Secure Partition Image returns the information to the UEFI.
      iii. The UEFI maps and marks the error record as the Runtime Services Data Region.
      iv. The error source information is updated in, or is added to, the ACPI Hardware Error Source Table (HEST).

2. The OS starts running.
3. By using an SMC call, the HEST driver scans the HEST table and registers error handlers for SDEI.
4. If a corrected (or an uncorrected error) occurs, the interrupt/exception will be routed to ATF in EL3.
5. ATF routes the event to the RAS error handler in the RAS firmware in S-EL1.
6. The RAS firmware creates the relevant ACPI CPER blobs and returns to ATF.
7. The ATF notifies the SDEI dispatcher to call the OS registered SDEI handler for uncorrected errors.

   Corrected errors and ACPI GSIV notifications are sent to the OS.
8. The OS SDEI callback (or ACPI GSIV handler for corrected errors) gets the ACPI CPER blobs by the Error Status Address block, processes the error, and tries to recover.
9. The error event is reported by using a RAS event in `/sys/kernel/debug/tracing`.
10. The `rasdaemon` logs the error information from the RAS event to the recorder.

Note: Errors can also be reported by using an ACPI Generic Event Device (GED), especially when the host OS does not register for SDEI.

SDEI Interrupts from Firmware First to the Arm CPU Core

The Arm SDEI is an architected way of delivering extraordinary system events from the CPU firmware to the kernel/hypervisor. Download the Software Delegated Exception Interface (SDEI) guide for more information.

As per SBBR guidance, only fatal/critical errors should be reported by using SDEI, and corrected errors should be reported by using the ACPI GED device. Host OSs that do not support or register SDEI will receive errors by using ACPI GED events notifications.

Errors

Memory Error Handling

The Linux kernel owns and manages the coherent memory, so memory errors must be reported to the kernel Memory Manager, and the Memory Manager can take appropriate recovery actions based on the affected pages. Grace Secure Firmware handles the CPU-attached LPDDR5 memory and Hopper GPU memory errors, and the errors are reported by using standard GHES flows with CPER memory error logs as defined in *Memory Error Section (N.2.5)* in the Unified Extensible Firmware Interface (UEFI) Specification version 2.9.

CPU loads to the uncorrected error Hopper chip memory, or the CPU chip memory might cause a SEA exception at the CPU, which must be handled by the CPU Firmware and reported to the kernel using the GHES SEA notification path. A CPER log of memory type that identifies the physical address of the memory error will be sent to the hypervisor.

Mapping for the UEFI Memory CPER Fields

The fields of a UEFI memory CPER follow a unique mapping to the memory on Grace.

Table 4. UEFI Memory CPER Mapping

CPER Field	Mapped Field
Node	Socket
ModuleRank	Device
Bank	Bank
Device	Channel
Row	Row
Column	Column
RequestorID	Requestor Client ID
RankNum	Device

Note: For HBM errors, the memory error section will have only physical address populated. The valid bits of the memory error will be 0 for all location fields (Node, Device, and so on).

Patrol Scrubbing (Background Scrubbing) of Grace LPDDR5 Memory

Grace RAS firmware will periodically use a hardware scrubber to scrub LPDDR memory.

If errors are detected because of the proactive scrubbing, the errors are reported to the hypervisor/host OS with the CPER memory log. Refer to "Memory Error Handling" on page 20 for more information.

The High-Speed Scrubber can be controlled using error injection. Refer to "Controlling HSS Using Error Injection" on Page 52.

CPU and CPU Cache Errors

For the CPU, CPU cache, and TLB errors, Firmware First creates a CPER log that includes an Arm Processor Error Section, as defined in *Arm Processor Error Section* (N.2.4.4) in the Unified Extensible Firmware Interface (UEFI) Specification version 2.9, and sends the log to the kernel with an SDEI callback.

Coherency NVLink Errors

RAS errors on the coherent NVLink between Grace sockets are also routed to the RAS firmware. The RAS firmware creates a CPER log with a Non-Standard Section Body as defined in Non-standard Section Body (N.2.3) of the Unified Extensible Firmware Interface (UEFI) Specification version 2.9. Refer to "CPER Decoding" on page 56 for more information about the Non-standard Section Body defined by NVIDIA. If it is a corrected error, the firmware sends an ACPI GED notification to the hypervisor. For fatal errors that cause a loss of coherency and Core lockup, the SBSA watchdog timer will cause an error recovery reset. The hardware retains the error state, and the Grace firmware provides the details about the NVLink error to the hypervisor and the BMC by using a BERT log on the reboot.

PCIe Errors

This section provides information about PCIe errors.

Advanced Error Reporting

For PCIe Advanced Error Reporting (AER) errors, the Firmware First entity must create a CPER log with the PCIe error section as defined in *PCI Express Error Section* (N.2.7) in the Unified Extensible Firmware Interface (UEFI) Specification version 2.9 and inject the log to the kernel.

Downstream Port Containment

Downstream Port Containment (DPC) errors will also be sent to Firmware First. When firmware owns the DPC, the firmware is expected to use the *Error Disconnect Recover* (EDR) notification to alert the OS-directed configuration and Power Management (OSPM) about a DPC event. If the OS supports EDR, the OSPM handles the software state invalidation and port recovery in the OS and makes an `_OST ACPI` call to notify the firmware about the recovery status. Refer to *Processing Sequence for Error Disconnect Recover* in the Advanced Configuration and Power Interface (ACPI) Specification, version 6.3 for more information about the related `_OST` status codes.

Also, as per *Downstream Port Containment Related Enhancements ECN* (section 4.5.1, table 4-6) in the PCI Firmware Specification Revision 3.2, if the DPC is controlled by the firmware (Firmware First mode), the firmware is responsible for configuring the DPC, and the OS is responsible for error recovery. The OS can modify DPC registers **only** during the EDR notification window, and with EDR support, the OS should provide DPC port services even in the Firmware First mode.

DPC is automatically disabled when a PCIe switch is connected to the root port with switch's downstream ports that support DPC. DPC is disabled for the root ports where GPUs are connected in the self-hosting mode.

Note When a PCIe error occurs that triggers DPC, the DPC and AER interrupt handlers are triggered for the same error condition. In the current implementation, each of these interrupt handlers generate two sets of CPERs (one for DPC and the second one for AER). This leads to generation of four CPERs, two coming from each handler.

UDPC Mechanism for PCI Express Uncorrectable Errors

When DPC is triggered and the hot-plug is disabled for the root port or when DPC is triggered and hot-plug is enabled but the device is not physically removed from the root port, the following workflow occurs:

1. The system is configured to raise DPC for certain types of Uncorrectable Non-Fatal/Fatal errors.

   When these errors occur, there will not be any CPERs that represent those errors sent to the BMC and OS because the errors were contained by the DPC. To avoid a kernel panic, CPERs that represent the Uncorrectable Non-Fatal/Fatal errors are not sent to the OS and BMC.
2. A CPER with the DPC information is sent to the BMC and OS.

   In the CPER, the severity is set to Corrected in the CPER because the error has already been managed by the DPC recovery mechanism.

3. A PCIe standard CPER with the AER error information is also sent to the OS and BMC, but with severity set to Corrected instead of Uncorrectable because the error was handled by DPC.
4. An Error Disconnect and Recovery (EDR) is sent to the OS from the firmware to inform the OS about the DPC event.
5. As part of the DPC handling mechanism, the OS reads the PCIe AER registers and the error information for the Completion Timeout error appears in the log.

When DPC is triggered, and the hot plug is enabled but the device is physically removed from the root port, the following workflow occurs:

1. A CPER with the DPC information is sent to the OS and the BMCA PCIe standard CPER with the AER error information is also sent to the OS and BMC.
2. A PCIe standard CPER with the AER error information is also sent to the OS and BMC, but with severity set to Corrected instead of Uncorrectable No CPER, and the DPC information is sent to the OS.
3. An EDR is not sent to the OS to inform OS about the DPC event.
4. The OS reads the PCIe AER registers and the error information for the Completion Timeout error appears in the log.

When DPC is disabled for the root port, the following workflow occurs:

1. A standard PCIE CPER is sent to the BMC and OS.
2. An NVIDIA-defined PCIe CPER will be sent to the BMC as an informational CPER.
3. The BMC logs will contain both CPERs but the dmesg logs will only display the standard PCIe CPER.

PCIe Root Complex Errors

Root Complex errors are non-standard, and RAS Firmware creates a CPER log with a Non Standard Section Body as defined in *Non-standard Section Body* (N.2.3) of the Unified Extensible Firmware Interface (UEFI) Specification version 2.9. Refer to "CPER Decoding" on page 56 for more information about the Non-standard Section Body as defined by NVIDIA. This log is sent to the kernel using an SDEI callback for uncorrected Root complex errors.

Note Starting in SBIOS version 02.03.04, PCIE CPERs with the PCIE-ANF IP signature were deprecated.

CPU-GPU NVLink Errors and GPU Containment

Errors on the Grace CPU to the GPU NVLink can cause CPU loads to coherent GPU memory to timeout, which causes the Grace hardware to enter the GPU containment mode.

Note: This containment mode can also be entered when a GPU is unresponsive.

Loads that are delivered to a contained GPU will return a **`Synchronous External Abort`** message to the CPU, so the hypervisor/kernel can precisely signal the process that delivered the load to the GPU memory. These types of errors require a reset and reinitialization of the GPU and the CPU-GPU NVLink. The errors are reported to the firmware, but these errors are shown to the hypervisor as a GPU PCIe AER internal error in CPER.

Instead of triggering a function-level reset of the GPU, these errors require that the hypervisor resets the GPU by invoking the `nvidia-smi --gpu_reset --id=ID` command. The firmware will now reset the GPU and retrain the NVlink, and after the GPU is reset, the hypervisor can start using the GPU memory again.

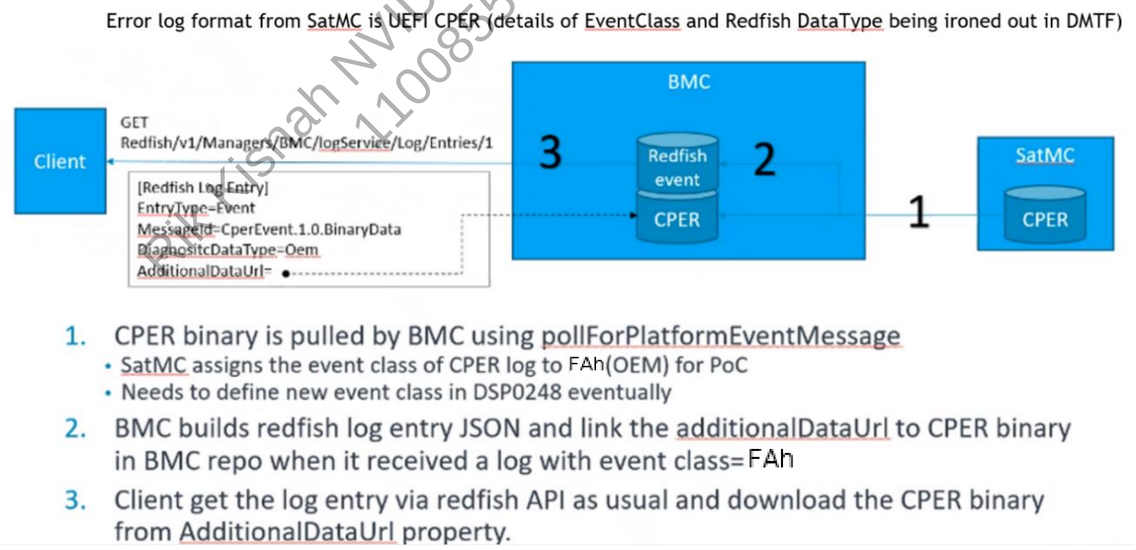
Logging RAS Error Logs to the BMC

All uncorrected hardware errors are reported to the BMC with the same CPER format as the errors that are sent to the OS. As per the *Arm Server Base Manageability Requirements 2.0* guide, here is the workflow to log a RAS error:

1. The RAS logs from the RAS firmware are sent to the BMC in the UEFI CPER format.
2. The RAS firmware notifies the BMC about a RAS Event by using Platform Level Data Model (PLDM) over Management Component Transport Protocol (MCTP) messages.

   If the CPER log fits in the buffer size that was registered by the BMC, the CPER is pushed to the BMC. Otherwise, a Platform event notification is sent to the BMC, and the BMC pulls the CPER log in parts until the entire log has been received. Refer to *section C.4.2* in the *Arm Server Base Manageability Requirements 2.0* guide for the detailed flow.
3. The BMC appends the CPER log to a Redfish event and sends the log to remote management clients.

Figure 6. Error Log Format



Error log format from SatMC is UEFI CPER (details of EventClass and Redfish DataType being ironed out in DMTF)

```
[Redfish Log Entry]
EntryType=Event
MessageId=CperEvent.1.0.BinaryData
DiagnositcDataType=Oem
AdditionalDataUrl=
```

1. CPER binary is pulled by BMC using pollForPlatformEventMessage
   - SatMC assigns the event class of CPER log to FAh(OEM) for PoC
   - Needs to define new event class in DSP0248 eventually
2. BMC builds redfish log entry JSON and link the additionalDataUrl to CPER binary in BMC repo when it received a log with event class=FAh
3. Client get the log entry via redfish API as usual and download the CPER binary from AdditionalDataUrl property.

Changing the CPER Destination

RAS firmware determines whether the CPER should be sent to the BMC, HEST, and/or BERT. Starting in SBIOS version 01.02.00, UEFI MM can change the destination of these CPERs by overriding the `RasHeader` flag. The `RasLogOverrideTargets` function in the `Silicon/NVIDIA/Drivers/SequentialRecordStMm/SequentialRecordComm.c` file provides an example where corrected CPERs are identified, and the CPER target destination is altered to prevent the CPERs from being sent to HEST.

The override function takes the following arguments:

> `RasPayload`: The complete CPER with an ACPI Generic Error Data Entry Structure header

> `Target`: A bitmap of targets that RAS Firmware selected and can be a combination of HEST, BERT, or BMC.

The function simply returns the Target that can be the same as the argument, or it can be altered by adding or removing targets.

Error Injection

Grace software supports correctable and uncorrectable error injection to the following error sources by using ACPI EINJ:

- Memory ECC
- CPU Complex injections
- PCIe injection
- Platform-specific error injection (NVIDIA-specific errors)

If the debugfs file system is not yet mounted, to mount it, run the following command:

```
mount -t debugfs none /sys/kernel/debug/
```

The error injection interface is available in the following directory:

```
cd /sys/kernel/debug/apei/einj
```

If the error injection interface is not present, to insert the einj kernel module, run the following command:

```
modprobe einj
```

Here is the standard procedure to inject an error type:

```
echo ERROR_TYPE > error_type
echo VENDOR_FLAG > vendor_flags    (Optional: Required only to inject Vendor specific
errors)
echo PARAM_1 > param1
echo 1 > error_inject
```

Note: You **must** enable a firmware configuration BCT that allows injection.

Note: The NVRASTool provides automated error injection for the errors mentioned in this section (NVOnline: **1112947**).

Note: Grace SBIOS does not support the kernel EINJ `notrigger` option, and it should remain set to 0.

Supported Error Types

The EINJ error types are in the `/sys/kernel/debug/apei/einj/available_error_type` directory:

```
root@localhost:/home/# cd /sys/kernel/debug/apei/einj
root@localhost:/sys/kernel/debug/apei/einj# cat available_error_type
0x00000001      Processor Correctable
0x00000004      Processor Uncorrectable fatal
0x00000008      Memory Correctable
0x00000010      Memory Uncorrectable non-fatal
0x00000040      PCI Express Correctable
0x00000080      PCI Express Uncorrectable non-fatal
0x00000100      PCI Express Uncorrectable fatal
```

Uncorrectable errors that do not trigger DPC are reported to the OS and BMC.

Correctable errors have different reporting mechanism based on the hardware in which the error occurs. The following sections provide more information about these reporting mechanisms.

In addition to the standard APEI defined error types. Grace also supports the following vendor-defined NVLink, PCI Express and Processor error types:

```
0x80800000      PCI Express Vendor Defined Error
0x84000000      Scalable Coherency Fabric Vendor Defined Error
0x80040000      GIC Node Vendor Defined Error
0x80020000      CPU NVLink Vendor Defined Error
0x90000000      NVLink C2C Vendor Defined Error
0xC0000000      DCC-ECC Vendor-defined error
```

Memory Error Injection

This section provides information about memory error injection.

ERROR_TYPES

- 0x00000008: Memory Correctable
- 0x00000010: Memory Uncorrectable non-fatal

Note: Memory correctable errors are silently corrected in transit and are never reported to the Host OS or BMC. Due to this error correction, `RAS_FW` will not be aware of a memory error, and a CPER will not be generated.

PARAM1

- For all memory errors, PARAM1 must be a valid 64-bit memory address.

PARAM2

For all memory errors, PARAM2 must be a valid physical memory address mask. To ensure that the memory is not used by `RAS_FW`, PARAM2 must be larger than 4K.

Note: The kernel will prevent error injection in memory that is not marked as System RAM and returns the `sh: write error: Invalid argument`.

To look for the system RAM, run the `cat /proc/iomem | grep "System RAM"` command.

Memory Uncorrectable Error Injection

Warning: Injecting uncorrected errors results in the offending page to be offlined and written to CMET (bad page list) with a special EINJ flag. On the following boot, the page will be removed from the memory map, and if the EINJ flag is set, the page is erased from CMET so that it does not persist.

Here is an example:

```
cd /sys/kernel/debug/apei/einj/
echo 0x00000010 > error_type
echo 0x100003000 > param1
echo 0xfffffffffffff000 > param2
echo 1 > error_inject
```

Here is the sample output:

```
Jan 16 17:04:41 localhost kernel: [  258.339831] EDAC MC0: 1 UE multi-bit ECC on unknown
memory (node:0 card:0 module:0 rank:0 bank:13 device:30 row:2184 column:512
requestor_id:0x00000000000000aa page:0x10000 offset:0x3000 grain:4096 - APEI location:
node:0 card:0 module:0 rank:0 bank:13 device:30 row:2184 column:512
requestor_id:0x00000000000000aa)
Jan 16 17:04:41 localhost kernel: [  258.339839] {1}[Hardware Error]: Hardware error from
APEI Generic Hardware Error Source: 308
Jan 16 17:04:41 localhost kernel: [  258.348493] {1}[Hardware Error]: event severity:
recoverable
Jan 16 17:04:41 localhost kernel: [  258.354362] {1}[Hardware Error]:  imprecise tstamp:
2024-01-16 17:04:42
Jan 16 17:04:41 localhost kernel: [  258.361209] {1}[Hardware Error]:  Error 0, type:
recoverable
Jan 16 17:04:41 localhost kernel: [  258.367079] {1}[Hardware Error]:   section_type:
memory error
Jan 16 17:04:41 localhost kernel: [  258.373038] {1}[Hardware Error]:   physical_address:
0x0000000100003000
```

```
Jan 16 17:04:41 localhost kernel: [  258.379885] {1}[Hardware Error]:
physical_address_mask: 0xfffffffffffff000
Jan 16 17:04:41 localhost kernel: [  258.387178] {1}[Hardware Error]:   node:0 card:0
module:0 rank:0 bank:13 device:30 row:2184 column:512 requestor_id:0x00000000000000aa
Jan 16 17:04:41 localhost kernel: [  258.399715] {1}[Hardware Error]:   error_type: 3,
multi-bit ECC
Jan 16 17:04:41 localhost kernel: [  258.406626] Memory failure: 0x10000: unhandlable
page.
Jan 16 17:04:41 localhost kernel: [  258.411936] Memory failure: 0x10000: recovery action
for unknown page: Ignored
```

Injecting an Error into the EGM Memory

When a portion of DRAM is assigned to EGM, it will not appear as regular DRAM to the OS. The OS checks the address of the injection to ensure it is correct, so you cannot use the standard memory error injection for EGM addresses. Use the following vendor-specific flow instead:

1. Using the 32-bit param1, set the upper part of the target address.
2. Using the 32-bit param1, set the lower part of the address and trigger the actual injection.

These steps must always be completed in the order in which they are listed.

ERROR_TYPE

0x80008000

PARAM1

For all memory errors, PARAM1 must be a valid 64-bit memory address.

1. In the first call, set the upper 32 bits of the address.

   No actual injection occurs.
2. In the second call, set the lower 32 bits of the address and trigger the injection.

Here is an example of an injection at address 0x1e6c000000, where the high part is 0x1E, and the low part is 0x6c000000:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80008000 > error_type
echo 0x1E > param1 #First call sets the upper 32 bits of the address.
echo 1 > vendor_flags
echo 1 > error_inject
sleep 0.5
echo 0x6c000000 > param1 #Second call sets the lower 32 bits of the address.
echo 1 > error_inject
```

PCI Express Error Injection

This section provides information about PCIe error injection.

Note: Errors should not be injected on a controller where the boot device is connected through a PCIE switch with other devices. Due to kernel limitations, if an endpoint cannot recover from the injected error, all devices, including the boot device, that are downstream from the PCIE switch will be offline.

Here is some additional information:

> Since the errors can be injected only in the root port, `Bus:Dev:Func` **always** needs to be zero.

  The `B:D:F` portion of `00:00:0` always represents the root port.

> The segment can change based on the root port in which errors will be injected.

> Ensure that you pay attention to this aspect in multi-socket systems.

ERROR_TYPES

> 0x00000040: PCI Express Correctable

> 0x00000080: PCI Express Uncorrectable non-fatal

> 0x00000100: PCI Express Uncorrectable fatal

> 0x80800000: Vendor-defined PCI Express Correctable

PARAM1

The Grace CPU RAS_FW uses the APEI EINJ PARAM1 reserved bits 0-7 to specify the error that will be injected for PCI Express. For all injecta:`ARAM_1` is constructed as follows:

```
 31              24 23          16 15             11 10                 8 7                             0
+-----------------+--------- ----+----------------+-------------------+------------------------------+
|     segment     |     bus      |     device     |     function      |     error_to_be_injected     |
+-----------------+----------- -+----------------+-------------------+------------------------------+
```

PCI Express Correctable Error Injection

This section provides information about PCI express correctable error injections.

Table 5. Injectable PCI Express Correctable Errors

Error Number	Error Description	Triggering Procedure
0x00	Receiver Error	None, error will be immediately reported
0x01	Bad DLLP	None, error will be immediately reported

Error Number	Error Description	Triggering Procedure
0x02	Bad TLP	None, error will be immediately reported
0x03	Replay Timer Timeout	None, error will be immediately reported
0x04	Reply Number Rollover	None, error will be immediately reported

Here is an example of injecting a PCI Express correctable receiver error:

```
cd /sys/kernel/debug/apei/einj/
echo 0x00000040 > error_type
echo 0x02000000 > param1
 for i in {1..33}; do echo "$i: ";echo 1 > error_inject; sleep 0.2; done
```

Here is the sample output:

```
{11}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 1281
{11}[Hardware Error]: It has been corrected by h/w and requires no further action
{11}[Hardware Error]: event severity: corrected
{11}[Hardware Error]:  imprecise tstamp: 2024-01-23 16:14:47
{11}[Hardware Error]:  Error 0, type: corrected
{11}[Hardware Error]:   section_type: PCIe error
{11}[Hardware Error]:   port_type: 4, root port
{11}[Hardware Error]:   command: 0x0507, status: 0x0011
{11}[Hardware Error]:   device_id: 0002:00:00.0
{11}[Hardware Error]:   slot: 0
{11}[Hardware Error]:   secondary_bus: 0x00
{11}[Hardware Error]:   vendor_id: 0x10de, device_id: 0x22b2
{11}[Hardware Error]:   class_code: 060400
{11}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0002
pcieport 0002:00:00.0: AER: aer_status: 0x00000001, aer_mask: 0x0000c000
pcieport 0002:00:00.0:    [ 0] RxErr                  (First)
pcieport 0002:00:00.0: AER: aer_layer=Physical Layer, aer_agent=Receiver ID
```

PCI Express Uncorrectable Non-Fatal Error Injection

Table 6. Injectable PCI Express Uncorrectable Non-Fatal Errors

Error Number	Error Description	Triggering Procedure
0x0f	Completion Timeout	To get the error injected and subsequently reported, try reading the endpoint's config space.

Here is an example of a PCI Express Completion Timeout uncorrectable non-fatal error:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x00000080 > error_type     # PCI Express Uncorrectable non-fatal
echo 0x0800000f > param1         # S:0x08,B:0x00,D:0x0,F:0x0,error_to_be_injected: 0x0f
echo 1 > error_inject            # Perform injection
```

To trigger the error injection, the endpoint device needs to be accessed. Reading the PCIE Endpoint configuration space is sufficient to trigger the error.

Here is the command to access the PCIE endpoint configuration space:

```
lspci -s 0008:04:00.0
```

Note

- In Grace systems, some of the Uncorrectable Non-Fatal errors (Including `Completion Timeout` error) trigger Downstream Port Containment (DPC).
- When injecting Completion Timeout errors into the system, Unexpected Completion (`UnxCmplt`) and Completion Timeout errors occur.

  This is because the way the error injection logic is implemented, it changes the `Function` number of the Requester-ID. The root port sees this change as an unexpected completion timeout and `UnxCmplt` is raised. Since the original completion was modified by the EINJ, the root port also raises a Completion Timeout because the port did not get the expected completion in time.

Here is the sample output:

```
[ 1983.122003] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 302
[ 1983.130641] {3}[Hardware Error]: event severity: recoverable
[ 1983.136425] {3}[Hardware Error]:  Error 0, type: recoverable
[ 1983.142208] {3}[Hardware Error]:   section_type: PCIe error
[ 1983.147902] {3}[Hardware Error]:   port_type: 4, root port
[ 1983.153508] {3}[Hardware Error]:   command: 0x0507, status: 0xc011
[ 1983.159824] {3}[Hardware Error]:   device_id: 0008:00:00.0
[ 1983.165430] {3}[Hardware Error]:   slot: 0
[ 1983.169614] {3}[Hardware Error]:   secondary_bus: 0x00
[ 1983.174863] {3}[Hardware Error]:   vendor_id: 0x10de, device_id: 0x22b9
[ 1983.181623] {3}[Hardware Error]:   class_code: 060400
[ 1983.186784] {3}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0002
[ 1983.194788] pcieport 0008:00:00.0: AER: aer_status: 0x00414000, aer_mask: 0x04500000
[ 1983.202718] pcieport 0008:00:00.0:    [14] CmpltTO
[ 1983.208955] pcieport 0008:00:00.0:    [16] UnxCmplt               (First)
[ 1983.215908] pcieport 0008:00:00.0: AER: aer_layer=Transaction Layer,
aer_agent=Requester ID
[ 1983.224457] pcieport 0008:00:00.0: AER: aer_uncor_severity: 0x00462030
[ 1983.231140] pcieport 0008:00:00.0: AER:   TLP Header: 4a000001 04000004 00010000
00000000
[ 1983.239567] ast 0008:04:00.0: AER: can't recover (no error_detected callback)
[ 1983.239580] xhci_hcd 0008:05:00.0: AER: can't recover (no error_detected callback)
[ 1983.239586] pcieport 0008:00:00.0: AER: device recovery failed
[ 1983.239637] {4}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 1281
[ 1983.239638] {4}[Hardware Error]: It has been corrected by h/w and requires no further
action
[ 1983.239640] {4}[Hardware Error]: event severity: corrected
[ 1983.239641] {4}[Hardware Error]:  Error 0, type: corrected
[ 1983.239642] {4}[Hardware Error]:   section_type: PCIe error
[ 1983.239643] {4}[Hardware Error]:   port_type: 4, root port
[ 1983.239644] {4}[Hardware Error]:   command: 0x0507, status: 0x0011
[ 1983.239645] {4}[Hardware Error]:   device_id: 0008:00:00.0
[ 1983.239646] {4}[Hardware Error]:   slot: 0
[ 1983.239647] {4}[Hardware Error]:   secondary_bus: 0x00
[ 1983.239648] {4}[Hardware Error]:   vendor_id: 0x10de, device_id: 0x22b9
[ 1983.239649] {4}[Hardware Error]:   class_code: 060400
[ 1983.239651] {4}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0002
[ 1983.239816] pcieport 0008:00:00.0: AER: aer_status: 0x0000a000, aer_mask: 0x0000e000
[ 1983.247750] pcieport 0008:00:00.0: AER: aer_layer=Transaction Layer,
aer_agent=Receiver ID
```

PCI Express Uncorrectable Fatal Error Injection

Table 7. Injectable PCI Express Uncorrectable Fatal Errors

Error Number	Error Description	Triggering Procedure
0x14	Data Link protocol Error	Nothing. When the error is injected, an error is reported.
0x15	Surprise Down Error	Nothing. When the error is injected, an error is reported.

Here is an example of an PCI Express Data Link Protocol uncorrectable fatal error:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x00000100 > error_type  # PCI Express Uncorrectable Fatal error
echo 0x02000014 > param1       # S:0x02,B:0x00,D:0x0,F:0x0, error_to_be_injected: 0x14
echo 1 > error_inject          # Perform injection
```

Note: In Grace systems, some of the Uncorrectable Fatal errors, such as the Data Link Protocol error, trigger DPC.

When DPC is disabled in the root port, to avoid an OS panic, a fatal error is reported as a recoverable, instead of a fatal, error to the OS.

Here is the sample output:

```
pcieport 0002:00:00.0: EDR: EDR event received
pcieport 0002:00:00.0: DPC: containment event, status:0x2009 source:0x0000
pcieport 0002:00:00.0: DPC: unmasked uncorrectable error detected
pcieport 0002:00:00.0: PCIe Bus Error: severity=Uncorrected (Fatal), type=Data Link
Layer, (Receiver ID)
pcieport 0002:00:00.0:   device [10de:22b2] error status/mask=00000010/04400000
pcieport 0002:00:00.0:    [ 4] DLP                    (First)
nvme nvme1: frozen state error detected, reset controller
nvme nvme1: restart after slot reset
pcieport 0002:00:00.0: pciehp: Slot(2): Link Down/Up ignored (recovered by DPC)
nvme nvme1: 64/0/0 default/read/poll queues
pcieport 0002:00:00.0: AER: device recovery successful
{17}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 1283
{17}[Hardware Error]: It has been corrected by h/w and requires no further action
```

```
{17}[Hardware Error]: event severity: corrected
{17}[Hardware Error]:  imprecise tstamp: 2024-01-23 16:39:17
{17}[Hardware Error]:  Error 0, type: corrected
{17}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-cb3fdb95c786
{17}[Hardware Error]:   section length: 0xf0
{17}[Hardware Error]:   00000000: 65494350 4350442d 00000000 00000000  PCIe-DPC........
{17}[Hardware Error]:   00000010: 00000000 00010002 140c0000 00000000  ................
{17}[Hardware Error]:   00000020: 00000002 80000000 000000c0 00000000  ................
{17}[Hardware Error]:   00000030: 0000000b 00000000 00000004 22b210de  ..............."
{17}[Hardware Error]:   00000040: 00060400 00000200 00000000 2981001d  ...............)
{17}[Hardware Error]:   00000050: 011a10c0 00002009 00000000 00000000  ..... ..........
{17}[Hardware Error]:   00000060: 00000000 00000000 00000000 00000000  ................
{17}[Hardware Error]:   00000070: 00000000 00000000 00000000 00000000  ................
{17}[Hardware Error]:   00000080: 00000000 00000000 00000000 15020001  ................
{17}[Hardware Error]:   00000090: 00000000 00400000 00462030 00000000  ......@.0 F.....
{17}[Hardware Error]:   000000a0: 0000e000 000000a0 00000000 00000000  ................
{17}[Hardware Error]:   000000b0: 00000000 00000000 00000000 00000000  ................
{17}[Hardware Error]:   000000c0: 00000000 00000000 00000000 00000000  ................
{17}[Hardware Error]:   000000d0: 00000000 00000000 00000000 1b810003  ................
{17}[Hardware Error]:   000000e0: 00000000 00000000 00000000 00000000  ................
{18}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 302
{18}[Hardware Error]: It has been corrected by h/w and requires no further action
{18}[Hardware Error]: event severity: corrected
{18}[Hardware Error]:  imprecise tstamp: 2024-01-23 16:39:17
{18}[Hardware Error]:  Error 0, type: corrected
{18}[Hardware Error]:   section_type: PCIe error
{18}[Hardware Error]:   port_type: 4, root port
{18}[Hardware Error]:   command: 0x0507, status: 0x0011
{18}[Hardware Error]:   device_id: 0002:00:00.0
{18}[Hardware Error]:   slot: 0
{18}[Hardware Error]:   secondary_bus: 0x00
{18}[Hardware Error]:   vendor_id: 0x10de, device_id: 0x22b2
{18}[Hardware Error]:   class_code: 060400
{18}[Hardware Error]:   bridge: secondary_status: 0x0000, control: 0x0002
pcieport 0002:00:00.0: AER: aer_status: 0x00000000, aer_mask: 0x0000c000
pcieport 0002:00:00.0: AER: aer_layer=Transaction Layer, aer_agent=Receiver ID
```

Vendor-Defined PCI Express Correctable Error Injection

When injecting a vendor-defined PCI Express error on Grace platforms, to signify that PARAM1 follows the `S:B:D.F` format for PCI Express errors, the `VENDOR_FLAG` must be set to 4.

Table 8. Vendor-Defined PCI Express Correctable Error Injection

Error Number	Error Description	Triggering Procedure
0x02	Error injection in Completion Re-order buffer	Try exercising the functionality of the device.

Error Number	Error Description	Triggering Procedure
		For an NVMe drive, try reading from, and writing to, the NVMe drive.
0x17	Error injection in Non-Posted Request FIFO	Try exercising the functionality of the device. For an NVMe drive, try reading from, and writing to, the NVMe drive.
0x19	Error injection in Posted Request FIFO	Try exercising the functionality of the device. For an NVMe drive, try reading from, and writing to, the NVMe drive.
0x1B	Error injection in Non-Posted Response FIFO	Try exercising the functionality of the device. For an NVMe drive, try reading from, and writing to, the NVMe drive.

Here are the examples for injecting a Vendor-defined PCI Express Non-Posted Request FIFO correctable error:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x80800000 > error_type     #Vendor-defined PCI Express correctable error
echo 4 > vendor_flags            #vendor flag set to 4 for PARAM1 as PCI Expresserror
echo 0x02000017 > param1         #S:0x02,B:0x00,D:0x0,F:0x0,error_to_be_injected:0x17
for i in {1..33}; do echo "$i: ";echo 1 > error_inject; dd if=/dev/nvme0n1 of=/dev/null
count=1;sleep 0.2; done
```

Note:

- The procedure to trigger the error injection involves exercising the functionality of the device, so the dd command is used every time `error_inject` is written with 1.
- Currently, you cannot exercise the functionality of the GPU to inject and report vendor-defined PCIe errors.

As a result, vendor-defined PCIE errors cannot be injected on a GPU, in a system containing an NVIDIA GPU.

Here is the sample output:

```
 [11780.416467] {2}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 1281
[11780.416477] {2}[Hardware Error]: event severity: info
[11780.416479] {2}[Hardware Error]:  imprecise tstamp: 2025-07-16 07:15:22
[11780.416481] {2}[Hardware Error]:  Error 0, type: info
[11780.416483] {2}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[11780.416486] {2}[Hardware Error]:   section length: 0xc0
[11780.416488] {2}[Hardware Error]:   00000000: 65494350 00000000 00000000 00000000
PCIe............
[11780.416490] {2}[Hardware Error]:   00000010: 00000000 000a0002 14120000 00000000
................
[11780.416492] {2}[Hardware Error]:   00000020: 14121010 00000000 49001502 00000000
..........I....
[11780.416494] {2}[Hardware Error]:   00000030: 14121014 00000000 00000000 00000000
```

```
................
[11780.416495] {2}[Hardware Error]:   00000040: 14121018 00000000 00000000 00000000
................
[11780.416497] {2}[Hardware Error]:   00000050: 1412101c 00000000 40000000 00000000
..........@....
[11780.416498] {2}[Hardware Error]:   00000060: 14123554 00000000 00000000 00000000
T5..............
[11780.416499] {2}[Hardware Error]:   00000070: 1412355c 00000000 00000000 00000000
\5..............
[11780.416500] {2}[Hardware Error]:   00000080: 14123564 00000000 00000000 00000000
d5..............
[11780.416502] {2}[Hardware Error]:   00000090: 1412356c 00000000 00000004 00000000
l5..............
[11780.416503] {2}[Hardware Error]:   000000a0: 14123574 00000000 00000000 00000000
t5..............
[11780.416504] {2}[Hardware Error]:   000000b0: 1412357c 00000000 00000000 00000000
|5..............
```

Standard APEI Processor Error Injection

This section provides information about processor error injection.

ERROR_TYPE

- 0x00000001: Processor Correctable
- 0x00000004: Processor Uncorrectable fatal

Processor Correctable Error Injection

Here is an example of injecting a Processor Correctable error:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 1 > error_type
echo 1 > error_inject
```

Note: The error type reported depends on the OS. In some situations, injecting TLB error might not result in a TLB error message.

Note: Standard processor correctable errors have a counter of size 127, and each error instance has its own counter. When the counter overflows, an error is reported, and RAS firmware resets the counter. Errors are only reported to the OS and BMC when the overflows.

Due to the way vendor-defined correctable errors are deduplicated and tracked in silicon, for injection testing, 129 correctable error injections might be necessary to correctly generate an error response. This is also the case when organic corrected errors are present before the injection test.

Here is the sample output:

```
[ 3150.322549] {1}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 516
[ 3150.322560] {1}[Hardware Error]: It has been corrected by h/w and requires no further
action
[ 3150.322563] {1}[Hardware Error]: event severity:
corrected                            "simowang-vm2" 21:52 16-Jun-25
[ 3150.322564] {1}[Hardware Error]: imprecise tstamp: 2025-06-17 04:54:42
[ 3150.322567] {1}[Hardware Error]: Error 0, type: corrected
[ 3150.322569] {1}[Hardware Error]:  section_type: ARM processor error
[ 3150.322571] {1}[Hardware Error]:  MIDR: 0x00000000410fd4f0
[ 3150.322573] {1}[Hardware Error]:  Multiprocessor Affinity Register (MPIDR):
0x0000000081020000
[ 3150.322575] {1}[Hardware Error]:  running state: 0x1
[ 3150.322577] {1}[Hardware Error]:  Power State Coordination Interface state: 0
[ 3150.322579] {1}[Hardware Error]:  Error info structure 0:
[ 3150.322580] {1}[Hardware Error]:  num errors: 1
[ 3150.322582] {1}[Hardware Error]:  error_type: 1, TLB error
[ 3150.322585] {1}[Hardware Error]:  error_info: 0x0000000004410015
[ 3150.322588] {1}[Hardware Error]:   transaction type: Data Access
[ 3150.322590] {1}[Hardware Error]:   TLB level: 1
[ 3150.322591] {1}[Hardware Error]:   the error has been corrected
[ 3150.322593] {1}[Hardware Error]:  virtual fault address: 0x9aa4420e0f6681fa
[ 3150.322596] {1}[Hardware Error]:  physical fault address: 0x80001315d636e5e5
[ 3150.322598] {1}[Hardware Error]:  Vendor specific error info has 64 bytes:
[ 3150.322601] {1}[Hardware Error]:  00000000: 3d594157 202c3020 42425553 3d4b4e41 WAY=
0, SUBBANK=
[ 3150.322604] {1}[Hardware Error]:  00000010: 42202c30 3d4b4e41 53202c30 52414255 0,
BANK=0, SUBAR
[ 3150.322606] {1}[Hardware Error]:  00000020: 3d594152 202c3020 45444e49 20203d58 RAY=
0, INDEX=
[ 3150.322609] {1}[Hardware Error]:  00000030: 202c3020 41525241 00303d59 00000000  0,
ARRAY=0.....
```

Processor Uncorrectable Fatal Error Injection

Here is an example of a core uncorrectable fatal error injection.

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 4 > error_type
echo 1 > error_inject
```

Here is the sample output:

```
 [ 3613.834995] {2}[Hardware Error]: event severity: fatal
:[ 3613.834999] {2}[Hardware Error]:  imprecise tstamp: 2024-01-17 20:20:17
 [ 3613.835002] {2}[Hardware Error]:  Error 0, type: fatal
0[ 3613.835005] {2}[Hardware Error]:   section_type: ARM processor error
x[ 3613.835007] {2}[Hardware Error]:   MIDR: 0x00000000410fd4f0
```

```
0[ 3613.835009] {2}[Hardware Error]:   Multiprocessor Affinity Register (MPIDR):
0x0000000081020000
0[ 3613.835011] {2}[Hardware Error]:   running state: 0x1
0[ 3613.835012] {2}[Hardware Error]:   Power State Coordination Interface state: 0
1[ 3613.835013] {2}[Hardware Error]:   Error info structure 0:
 [ 3613.835014] {2}[Hardware Error]:   num errors: 1
 [ 3613.835015] {2}[Hardware Error]:    error_type: 0, cache error
 [ 3613.83V5016] {2}[Hardware Error]:    error_info: 0x0000000000410015
M[ 3613.835018] {2}[Hardware Error]:     transaction type: Data Access
 [ 3613.835019] {2}[Hardware Error]:     cache level: 1
8[ 3613.835020] {2}[Hardware Error]:     the error has not been corrected
0[ 3613.835020] {2}[Hardware Error]:    virtual fault address: 0xde151c06a05c8c3a
0[ 3613.835021] {2}[Hardware Error]:    physical fault address: 0x80001c300ca44193
1[ 3613.835021] {2}[Hardware Error]:   Vendor specific error info has 64 bytes:
:[ 3613.835025] {2}[Hardware Error]:    00000000: 3d594157 202c3020 42425553 3d4b4e41
WAY= 0, SUBBANK=
 [ 3613.835025] {2}[Hardware Error]:    00000010: 42202c30 3d4b4e41 53202c30 52414255  0,
BANK=0, SUBAR
I[ 3613.835026] {2}[Hardware Error]:    00000020: 3d594152 202c3020 45444e49 20203d58
RAY= 0, INDEX=
N[ 3613.835027] {2}[Hardware Error]:    00000030: 202c3020 41525241 00303d59 00000000
0, ARRAY=0.....
```

Hardware Safety Manager (HSM) Error Injection

This section provides information about vendor-defined HSM error injection.

ERROR_TYPE

> 0x81000000: Hardware Safety Manager (HSM) Vendor-defined error

VENDOR_FLAG

For vendor-defined processor errors, the vendor flag must always be set to 1 and match the PARAM1 format for processor errors.

PARAM1

Table 9. Param1 Structure for HSM EINJ

Bit(s)	Field	Note
31	L2 Reset	Setting this bit will cause a system reboot. A BERT Record will be generated on the next boot.

Bit(s)	Field	Note
30	Sub-index Flag	Inject an error in the source defined by Sub-Index/Index (12-19)
29	Error Collator Flag	Inject an error in the error collator. Bit 30 must also be set, and subindex must match the error collator. CPER generation is not guaranteed
28	Status Register Scan	Scan all status registers and verify they are accessible. A CPER will not be generated for this action.
27-24	Reserved	
23-20	Socket	Sockets 0-3.
19-12	Sub-Index	Error source within HSM index. Not all sub-index/index combinations are valid. CPER generation is not guaranteed.
11-0	Index	HSM Error Source Number. Not all indexes are valid. CPER generation is not guaranteed.

Here is an example of a vendor-defined HSM error injection.

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x81000000 > error_type
echo 0x600040B8 > param1
echo 0x1> vendor_flags
echo 1 > error_inject
```

Here is the sample output:

```
[ 1149.412047] {1}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 2561
[ 1149.412053] {1}[Hardware Error]: It has been corrected by h/w and requires no further
action
[ 1149.412056] {1}[Hardware Error]: event severity: corrected
[ 1149.412058] {1}[Hardware Error]:  imprecise tstamp: 2024-04-08 22:04:46
[ 1149.412062] {1}[Hardware Error]:  Error 0, type: corrected
[ 1149.412064] {1}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[ 1149.412067] {1}[Hardware Error]:   section length: 0x30
[ 1149.412070] {1}[Hardware Error]:   00000000: 004d5348 00000000 00000000 00000000
HSM............
[ 1149.412072] {1}[Hardware Error]:   00000010: 00000004 00010000 04040000 00000000
...............
[ 1149.412074] {1}[Hardware Error]:   00000020: 0404d0d8 00000000 00000020 00000000
........
```

DCC-ECC Error Injection

This section has information on vendor-defined DCC-ECC error injection.

ERROR_TYPE

> 0xC0000000 DCC-ECC Vendor-defined error

VENDOR_FLAG

For vendor-defined processor errors, the vendor flag must always be set to 1 and match the PARAM1 format for processor errors.

PARAM1

Table 10. Param1 Structure for DCC-ECC EINJ

Bit(s)	Field	Note
31-28	Reserved	
27-24	Bitmap of sockets	• Bit 27: Socket 3 • Bit 26: Socket 2 • Bit 25: Socket 1 • Bit 24: Socket 0
23-8	Bitmap of Level 2 cache(LTC) slices	• 8 LTC [0-7] with 2 slices each [0-1] • Bit 23: LTC7/Slice1 • Bit 22: LTC7/Slice0 • Bit 21: LTC6/Slice1 • Bit 20: LTC6/Slice0 • Bit 19: LTC5/Slice1 • Bit 18: LTC5/Slice0 • Bit 17: LTC4/Slice1 • Bit 16: LTC4/Slice0 • Bit 15: LTC3/Slice1 • Bit 14: LTC3/Slice0 • Bit 13: LTC2/Slice1 • Bit 12: LTC2/Slice0 • Bit 11: LTC1/Slice1 • Bit 10: LTC1/Slice0 • Bit 9: LTC0/Slice1 • Bit 8: LTC0/Slice0
7-2	Reserved	

Bit(s)	Field	Note
1	L2 Reset	• 1: Trigger L2 Reset • 0: Normal
0	Severity	• 1: Uncorrected ECC Error • 0: Corrected ECC Error

Here is an example of a vendor defined DCC-ECC error injection.

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0xC0000000 > error_type
echo 0x01010100 > param1
echo 0x1> vendor_flags
echo 1 > error_inject
```

Here is the sample output:

```
[ 17281.295483] {2}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 801
[17281.295488] {2}[Hardware Error]: It has been corrected by h/w and requires no further
action
[17281.295490] {2}[Hardware Error]: event severity: corrected
[17281.295493] {2}[Hardware Error]:  imprecise tstamp: 2024-04-09 02:33:38
[17281.295495] {2}[Hardware Error]:  Error 0, type: corrected
[17281.295497] {2}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[17281.295499] {2}[Hardware Error]:   section length: 0x60
[17281.295503] {2}[Hardware Error]:   00000000: 2d434344 00434345 00000000 00000000  DCC-
ECC.........
[17281.295505] {2}[Hardware Error]:   00000010: 00000000 00040102 04e10400 00000000
................
[17281.295507] {2}[Hardware Error]:   00000020: 04e104f0 00000000 00000002 00000000
................
[17281.295508] {2}[Hardware Error]:   00000030: 04e104f8 00000000 00000000 00000000
................
[17281.295510] {2}[Hardware Error]:   00000040: 04e104f4 00000000 00010001 00000000
................
[17281.295512] {2}[Hardware Error]:   00000050: 04e104fc 00000000 4ab68993 00000000
..........J....
[17281.295514] {2}[Hardware Error]:  imprecise tstamp: 2024-04-09 02:33:38
[17281.295515] {2}[Hardware Error]:  Error 1, type: corrected
[17281.295517] {2}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[17281.295518] {2}[Hardware Error]:   section length: 0x60
[17281.295520] {2}[Hardware Error]:   00000000: 2d434344 00434345 00000000 00000000  DCC-
ECC.........
[17281.295521] {2}[Hardware Error]:   00000010: 00000000 00040102 04e50400 00000000
................
[17281.295523] {2}[Hardware Error]:   00000020: 04e504f0 00000000 00000002 00000000
................
[17281.295524] {2}[Hardware Error]:   00000030: 04e504f8 00000000 00000000 00000000
................
[17281.295526] {2}[Hardware Error]:   00000040: 04e504f4 00000000 00010001 00000000
................
```

```
[17281.295528] {2}[Hardware Error]:   00000050: 04e504fc 00000000 40000016 00000000
...........@....
```

Vendor-Defined Processor Error Injection

This section has information on vendor-defined processor error injection.

ERROR_TYPE

> 0x84000000: Processor Scalable Coherency Fabric (SCF) Node Vendor-defined error.
> 0x80040000: Processor Generic Interrupt Controller (GIC) Node Vendor-defined error.

VENDOR_FLAG

For vendor-defined processor errors, the vendor flag must always be set to 1 and match the PARAM1 format for processor errors.

PARAM1

Table 11. Param1 Structure for SCF EINJ

Bit(s)	Field	Note
31-30	Socket	Sockets 0-3
29	Reserved	Reserved
28	L2 Reset	When set, triggers an L2 reset after injecting the error.
27-24	SCF Type	• 0: SCC • 1: IOB • 2: MCF_BRIDGE • 3: SOC_BRIDGE • 4: CCPMU • 5: CSN • 6: ABU • 7: CMU_CLOCKS • 8: CLUSTER
23-16	SCF Instance	• 0-167: SCC • 0: IOB • 0-7: MCF_BRIDGE • 0-3: SOC_BRIDGE

Bit(s)	Field	Note
		• 0: CCPMU • 0-41: CSN • 0: ABU • 0: CMU_CLOCKS • 0-255: CLUSTER
15	Severity	• 0: Correctable Error. • 1: Uncorrectable Error.
14-8	Reserved	
7-0	Error Instance	32-63. Error instances depend on the SCF Type and Instance. Refer to Table 12 for the error instance options.

Table 12. Supported SCF Error Instances

Unit	Correctable Error Instances	Uncorrectable Error Instances
SCC	0x20: CTAG ECC single bit corrected. 0x21: MDIR ECC single bit corrected. 0x22: Data ECC single bit corrected.	0x2c: Data ECC DED Uncorrectable. 0x29: SCC Lockable CREG Error.
IOB	0x20: The local IOB destination write data packet has an ECC correctable error.	0x33: IOB CRAB node was received posted write access to a locked CREG.
MCF_BRIDGE	Not Supported.	0x20: Lockable CREG error from the RAS CREG node.
SOC_BRIDGE	Not Supported.	0x20: Lockable CREG error from the RAS CREG node.
CCPMU	Not Supported.	0x21: CCPMU unit CREG node lock error.
CSN	Not Supported.	0x21: CSN Lockable CREG Error.
ABU	0x20: DVMU Correctable ECC. 0x21: Correctable ECC.	0x29: CMULA Lockable Error.
CMU_CLOCKS	Not Supported.	0x20: Error Generated by Lock Error.
CLUSTER	0x20: A cluster lockable CREG is already locked.	0x25: SCF ECC Data SEC.
GIC	N/A	N/A

Table 13. Param1 Structure for GIC EINJ

Bit(s)	Field	Note
31-30	Socket	Sockets 0-3
29-4	Reserved	Reserved
3-0	GIC instance	Must be 0x0 – 0x3

SCF Error Injection

Note: SCF nodes that support correctable errors have a counter of size 127, and each error instance has its own counter. When the counter overflows, the error is reported, and RAS firmware resets the counter. Errors are only reported to the OS and BMC when the overflows.

Due to the way vendor-defined correctable errors are deduplicated and tracked in silicon, for injection testing, 129 correctable error injections might be necessary to correctly generate an error response. This is also the case when organic corrected errors are present before the injection test.

If errors are injected into one type of instance, and then injected into a different type of instance, a CPER will be reported for the first type of instance followed by a CPER that identified the error for the second type of instance, and the CPER that was injected that caused the counter to overflow.

Here is an example of injecting SCF error:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x84000000 > error_type          # SCF error type
echo 1 > vendor_flags                 # vendor flag set for processor errors
echo 0x00038029 > param1              # SCC Lockable CREG error on socket 0 SCC 0x39
echo 1 > error_inject
```

Here is the sample output:

```
 [ 1602.246759] {1}[Hardware Error]: event severity: fatal
O[ 1602.246761] {1}[Hardware Error]:  imprecise tstamp: 2024-01-16 22:01:50
:[ 1602.246762] {1}[Hardware Error]:  Error 0, type: fatal
 [ 1602.246764] {1}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
 [ 1602.246766] {1}[Hardware Error]:   section length: 0x90
 [ 1602.246768] {1}[Hardware Error]:  00000000: 4c504343 43535845 00000046 00000000
CCPLEXSCF.......
 [ 1602.246769] {1}[Hardware Error]:  00000010: 00000000 00060001 1820d000 00000000
.......... .....
C[ 1602.246770] {1}[Hardware Error]:  00000020: 1820d010 00000000 6400090e 00000000 ..
........d....
P[ 1602.246771] {1}[Hardware Error]:  00000030: 00000000 00000000 00000000 00000000
................
E[ 1602.246772] {1}[Hardware Error]:  00000040: 1820d020 00000000 00000000 00000000  .
.............
R[ 1602.246773] {1}[Hardware Error]:  00000050: 1820d028 00000000 00000000 00000000 (.
.............
```

```
 [ 1602.246774] {1}[Hardware Error]:  00000060: 1820d030 00000000 00000000 00000000 0.
.............
H[ 1602.246775] {1}[Hardware Error]:  00000070: 1820d038 00000000 00000000 00000000 8.
.............
e[ 1602.246775] {1}[Hardware Error]:  00000080: 00000000 00000000 00000000 00000000
................
a[ 1602.246776] {1}[Hardware Error]:  imprecise tstamp: 2024-01-16 22:01:50
d[ 1602.246777] {1}[Hardware Error]:  Error 1, type: fatal
e[ 1602.246778] {1}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
r[ 1602.246778] {1}[Hardware Error]:   section length: 0x90
 [ 1602.246779] {1}[Hardware Error]:  00000000: 4c504343 43535845 00000046 00000000
CCPLEXSCF.......
 [ 1602.246780] {1}[Hardware Error]:  00000010: 00000000 00060001 1830d000 00000000
..........0.....
V[ 1602.246781] {1}[Hardware Error]:  00000020: 1830d010 00000000 6400090e 00000000
..0........d....
M[ 1602.246782] {1}[Hardware Error]:  00000030: 00000000 00000000 00000000 00000000
................
 [ 1602.246782] {1}[Hardware Error]:  00000040: 1830d020 00000000 00000000 00000000
.0.............
8[ 1602.246783] {1}[Hardware Error]:  00000050: 1830d028 00000000 00000000 00000000
(.0.............
0[ 1602.246784] {1}[Hardware Error]:  00000060: 1830d030 00000000 00000000 00000000
0.0.............
0[ 1602.246784] {1}[Hardware Error]:  00000070: 1830d038 00000000 00000000 00000000
8.0.............
1[ 1602.246785] {1}[Hardware Error]:  00000080: 00000000 00000000 00000000
00000000................
```

GIC Error Injection

Here is an example of GIC Error Injection:

```
modprobe einj
cd /sys/kernel/debug/apei/einj/
echo 0x80040000 > error_type
echo 1 > vendor_flags
echo 0 > param1
echo 1 > error_inject
```

NVLink Error Injection

This section provides information about NVLink error injection.

ERROR_TYPE

- 0x90000000: Vendor-defined Uncorrectable NVLink-C2C error
- 0x80020000: Vendor-defined Uncorrectable CPU NVLink error type

VENDOR_FLAG

For NVLink errors, the vendor flag must always be set to 1 and match the PARAM1 format for processor errors.

NVLink-C2C Error Injection

⚠ **Warning:** Here are some additional warnings:

- Injecting a NVLINK-C2C error with the L2 reset bit set in Param1, will cause RAS_FW to initiate a system reboot.
- An L2 reset is always going to be triggered when injecting a NVLINK-C2C fatal error on a system where C2C is used between two Grace CPUs.

PARAM1

Table 14. Param1 Structure for NVLink-C2C EINJ

Bit(s)	Field	Note
31-30	Socket	Sockets 0-3
29	Reserved	Reserved
28	L2 Reset	If set, will trigger an L2 reset after injecting the error
27-24	C2C Instance	C2C [0-9]
23	Error Injection Method	1: Using DEBUG INTR
5-0	Reserved	N/A

This section provides information about NVLink-C2C errors.

```
modprobe einj                           # Load EINJ driver (if not already loaded)
cd /sys/kernel/debug/apei/einj          # Access APEI EINJ interface
echo 0x90000000 > error_type            # C2C error
echo 0x01 > vendor_flags                # vendor flag set to 1 for processor error
echo 0x00800000 > param1                # CPU Socket 0, C2C Link 0
echo 1 > error_inject                   #perform injection
```

Here is the sample output:

```
[  648.570573] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 2817
[  648.579372] {3}[Hardware Error]: event severity: recoverable
[  648.585240] {3}[Hardware Error]:  imprecise tstamp: 2024-04-25 19:39:01
[  648.592086] {3}[Hardware Error]:  Error 0, type: fatal
[  648.597424] {3}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[  648.606317] {3}[Hardware Error]:   section length: 0x200
[  648.611841] {3}[Hardware Error]:   00000000: 00433243 00000000 00000000 00000000
C2C.............
[  648.621085] {3}[Hardware Error]:   00000010: 00000000 001e0101 13fe2000 00000000
......... ......
[  648.630334] {3}[Hardware Error]:   00000020: 13fe2264 00000000 00000000 00000000
d"..............
[  648.639582] {3}[Hardware Error]:   00000030: 13fe3264 00000000 00000000 00000000
d2..............
[  648.648831] {3}[Hardware Error]:   00000040: 13fe4264 00000000 00000000 00000000
dB..............
[  648.658081] {3}[Hardware Error]:   00000050: 13fe5264 00000000 00000000 00000000
dR..............
[  648.667330] {3}[Hardware Error]:   00000060: 13fe6264 00000000 00000000 00000000
db..............
[  648.676580] {3}[Hardware Error]:   00000070: 13fe7264 00000000 00000000 00000000
dr..............
[  648.685828] {3}[Hardware Error]:   00000080: 13fe8264 00000000 00000000 00000000
d...............
[  648.695077] {3}[Hardware Error]:   00000090: 13fe9264 00000000 00000000 00000000
d...............
[  648.704326] {3}[Hardware Error]:   000000a0: 13fea264 00000000 00000000 00000000
d...............
[  648.713575] {3}[Hardware Error]:   000000b0: 13feb264 00000000 00000000 00000000
d...............
[  648.722824] {3}[Hardware Error]:   000000c0: 13fe2430 00000000 00000000 00000000
0$..............
[  648.732073] {3}[Hardware Error]:   000000d0: 13fe3430 00000000 00000000 00000000
04..............
[  648.741322] {3}[Hardware Error]:   000000e0: 13fe4430 00000000 00000000 00000000
0D..............
[  648.750572] {3}[Hardware Error]:   000000f0: 13fe5430 00000000 00000000 00000000
0T..............
[  648.759821] {3}[Hardware Error]:   00000100: 13fe6430 00000000 00000000 00000000
0d..............
[  648.769070] {3}[Hardware Error]:   00000110: 13fe7430 00000000 00000000 00000000
0t..............
[  648.778320] {3}[Hardware Error]:   00000120: 13fe8430 00000000 00000000 00000000
0...............
[  648.787568] {3}[Hardware Error]:   00000130: 13fe9430 00000000 00000000 00000000
0...............
[  648.796818] {3}[Hardware Error]:   00000140: 13fea430 00000000 00000000 00000000
0...............
[  648.805992] {3}[Hardware Error]:   00000150: 13feb430 00000000 00000000 00000000
0...............
[  648.815241] {3}[Hardware Error]:   00000160: 13fe2240 00000000 00000002 00000000
@"..............
[  648.824491] {3}[Hardware Error]:   00000170: 13fe3240 00000000 00000000 00000000
@2..............
[  648.833740] {3}[Hardware Error]:   00000180: 13fe4240 00000000 00000000 00000000
@B..............
[  648.842988] {3}[Hardware Error]:   00000190: 13fe5240 00000000 00000000 00000000
@R..............
[  648.852237] {3}[Hardware Error]:   000001a0: 13fe6240 00000000 00000000 00000000
@b..............
```

```
[  648.861486] {3}[Hardware Error]:   000001b0: 13fe7240 00000000 00000000 00000000
@r..............
[  648.870736] {3}[Hardware Error]:   000001c0: 13fe8240 00000000 00000000 00000000
@...............
[  648.879985] {3}[Hardware Error]:   000001d0: 13fe9240 00000000 00000000 00000000
@...............
[  648.889234] {3}[Hardware Error]:   000001e0: 13fea240 00000000 00000000 00000000
@...............
[  648.898483] {3}[Hardware Error]:   000001f0: 13feb240 00000000 00000000 00000000
@...............
```

CPU NVLink Error Injection

Note: CPU NVLink errors can only occur and be injected on systems that have at least two GH200 modules, and the CPUs are connected by CPU NVLinks.

Warning: Here are some additional warnings:

- An L2 reset is always going to be triggered when you inject a CPU NVLink Uncorrectable fatal error.
- Injecting a CPU NVLink error with the L2 reset bit set in Param1 will cause RAS_FW to initiate a system reboot.

PARAM1

Table 15. Param1 Structure for CPU NVLink EINJ

Bit(s)	Field	Note
31-30	Socket	Sockets 0-3
29	NVLW instance	NVLW instance: 0-1, as there are two NVLW instances per socket
28	Reserved	
27-26	Severity	2: Uncorrectable non-fatal 3: Uncorrectable fatal
25	L2 Reset	If set, it will trigger an L2 reset after injecting the error
24-0	Reserved	

Here is an example of a vendor-defined Uncorrectable CPU NVLink error injection:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80020000 > error_type
echo 0xC000000  > param1
echo 1 > vendor_flags
echo 1 > error_inject
```

An uncorrectable CPU NVLink error will immediately cause a system reboot, and as a result, there is no output in the console when injecting this error.

SOCHUB

This section provides information about injecting SOCHUB errors.

Note: This error is not injectable in hardware, but the RAS firmware will read the appropriate registers and simulate values based on selected parameters.

ERROR_TYPE

> 0x8040 0000

VENDOR_FLAG

For SOCHUB errors, the vendor flag must always be set to 1.

PARAM1

Table 16. PARAM1

Bit(s)	Value	Notes
0-4	0-3	Socket
8-15	0-7	MCF Slice
16-24		Client CGID that consumed poison
24-31	0	Single Error
	1	Multi Error

For example, to simulate a poison consumed by read from PCIe7 client (CGID=0x2A) on socket 0 and aperture 2, which is M2 for this client:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80400000  > error_type
echo 0x012A0200 > param1
echo 1 > vendor_flags
echo 1 > error_inject
```

Here is the sample output:

```
localhost:/sys/kernel/debug/apei/einj# [487027.394133] {3}[Hardware Error]: Hardware
error from APEI Generic Hardware Error Source: 832
[487027.407954] {3}[Hardware Error]: event severity: fatal
[487027.418147] {3}[Hardware Error]:  Error 0, type: fatal
[487027.428237] {3}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[487027.441884] {3}[Hardware Error]:   section length: 0x140
[487027.452048] {3}[Hardware Error]:   00000000: 48434f53 00004255 00000000 00000000
SOCHUB..........
[487027.470469] {3}[Hardware Error]:   00000010: 00000000 00120001 04020000 00000000
................
[487027.488856] {3}[Hardware Error]:   00000020: 0402600c 00000000 00000008 00000000
.`..............
[487027.507215] {3}[Hardware Error]:   00000030: 04026028 00000000 00000000 00000000
(`..............
[487027.525537] {3}[Hardware Error]:   00000040: 0404600c 00000000 00000000 00000000
.`..............
[487027.543818] {3}[Hardware Error]:   00000050: 04046028 00000000 00000000 00000000
(`..............
[487027.562073] {3}[Hardware Error]:   00000060: 0406600c 00000000 00000000 00000000
.`..............
[487027.580298] {3}[Hardware Error]:   00000070: 04066028 00000000 00000000 00000000
(`..............
[487027.598487] {3}[Hardware Error]:   00000080: 0408600c 00000000 00000000 00000000
.`..............
[487027.616642] {3}[Hardware Error]:   00000090: 04086028 00000000 8000002a 00000000
(`......*.......
[487027.634769] {3}[Hardware Error]:   000000a0: 040a600c 00000000 00000000 00000000
.`..............
[487027.652861] {3}[Hardware Error]:   000000b0: 040a6028 00000000 00000000 00000000
(`..............
[487027.670912] {3}[Hardware Error]:   000000c0: 040c600c 00000000 00000000 00000000
.`..............
[487027.688937] {3}[Hardware Error]:   000000d0: 040c6028 00000000 00000000 00000000
(`..............
[487027.706931] {3}[Hardware Error]:   000000e0: 040e600c 00000000 00000000 00000000
.`..............
[487027.724887] {3}[Hardware Error]:   000000f0: 040e6028 00000000 00000000 00000000
(`..............
[487027.742817] {3}[Hardware Error]:   00000100: 0410600c 00000000 00000000 00000000
.`..............
[487027.760934] {3}[Hardware Error]:   00000110: 04106028 00000000 00000000 00000000
(`..............
[487027.779288] {3}[Hardware Error]:   00000120: 0412600c 00000000 00000000 00000000
.`..............
[487027.797767] {3}[Hardware Error]:   00000130: 04126028 00000000 00000000 00000000
(`..............
```

Controlling HSS Using Error Injection

This section describes how HSS (patrol scrubbing) can be controlled using error injection.

ERROR_TYPE

> 0x80000001

VENDOR_FLAG

For this error injection, the vendor flag must always be set to 1.

PARAM1

Table 17. Param 1 structure for HSS EINJ

Bit(s)	Value	Notes
0-7	0	Request status (sends informational CPER)
	1	Request to pause/flush on all HSS instances
	2	Start Patrol Scrubbing

Here is an example of pausing/flushing the HSS:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80000001 > error_type
echo 1 > param1
echo 1 > vendor_flags
echo 1 > error_inject
```

Controlling PCIe Leaky Buckets Using Error Injection

This section provides information about controlling the Leaky Buckets for PCIe using error injection.

ERROR_TYPE

> 0x8000 0040

VENDOR_FLAG

For this error injection, the vendor flag must always be set to 1.

PARAM1

Table 18 PARAM1

Bits	Values
0-8	1: Reset all leaky buckets (as if no previous corrected errors were received) 2: Disable all leaky buckets (i.e. each corrected error will be reported right away) 3: Enable all leaky buckets

Here is an example of resetting all the leaky buckets:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80000040 > error_type
echo 1 > param1
echo 1 > vendor_flags
echo 1 > error_inject
```

CMET-INFO

This section provides information about CMET-INFO error injection. CMET-INFO CPER outputs the status of the CMET.

ERROR_TYPE

> 0x80000010

PARAM1

For this error injection, use PARAM1 as 3.

VENDOR_FLAG

For this error injection, the vendor flag must always be set to 1.

Here is an example of injecting the CMET-INFO CPER:

```
cd /sys/kernel/debug/apei/einj/
echo 0x80000010 > error_type
echo 3 > param1
echo 1 > vendor_flags
echo 1 > error_inject
```

Here is a sample output:

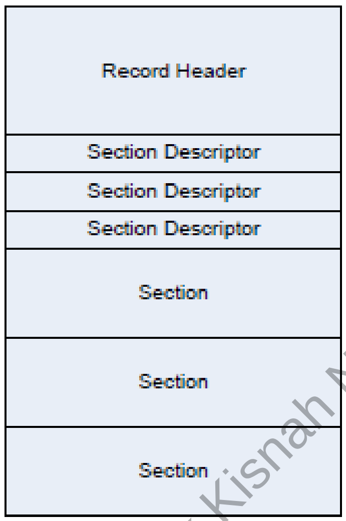
```
[62274.887628] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error
Source: 896
[62274.887688] {3}[Hardware Error]: event severity: info
[62274.887722] {3}[Hardware Error]: imprecise tstamp: 2024-05-26 15:42:46
[62274.887767] {3}[Hardware Error]: Error 0, type: info
[62274.887795] {3}[Hardware Error]: section type: unknown, 6d5244f2-2712-11ec-bea7-
cb3fdb95c786
[62274.887830] {3}[Hardware Error]: section length: 0x220
[62274.887873] {3}[Hardware Error]: 00000000: 54454d43 464e492d 0000004f 00000000 CMET-
INFO.......
[62274.887911] {3}[Hardware Error]: 00000010: 00000000 00200103 00000000 00000000 ......
.........
[62274.887946] {3}[Hardware Error]: 00000020: 00000000 80001001 00000000 00000001
................
[62274.887979] {3}[Hardware Error]: 00000030: 00000001 80001001 00000000 00000001
................
[62274.888008] {3}[Hardware Error]: 00000040: 00000002 80001001 00000000 00000001
................
[62274.888034] {3}[Hardware Error]: 00000050: 00000003 80001001 00000000 00000001
................
[62274.888061] {3}[Hardware Error]: 00000060: 00000004 80001001 00000000 00000001
................
[62274.888088] {3}[Hardware Error]: 00000070: 00000005 80001001 00000000 00000001
................
[62274.888114] {3}[Hardware Error]: 00000080: 00000006 80001001 00000000 00000001
................
[62274.888141] {3}[Hardware Error]: 00000090: 00000007 80001001 00000000 00000001
................
[62274.888168] {3}[Hardware Error]: 000000a0: 00000008 80001001 00000000 00000001
................
[62274.888193] {3}[Hardware Error]: 000000b0: 00000009 80001001 00000000 00000001
................
[62274.888219] {3}[Hardware Error]: 000000c0: 0000000a 80001001 00000000 00000001
................
[62274.888245] {3}[Hardware Error]: 000000d0: 0000000b 80001001 00000000 00000001
................
[62274.888273] {3}[Hardware Error]: 000000e0: 0000000c 80001001 00000000 00000001
................
[62274.888299] {3}[Hardware Error]: 000000f0: 0000000d 80001001 00000000 00000001
................
[62274.888325] {3}[Hardware Error]: 00000100: 0000000e 80001001 00000000 00000001
................
[62274.888355] {3}[Hardware Error]: 00000110: 0000000f 80001001 00000000 00000001
................
[62274.888383] {3}[Hardware Error]: 00000120: 00000010 80001001 00000000 00000001
................
[62274.888410] {3}[Hardware Error]: 00000130: 00000011 80001001 00000000 00000001
................
[62274.888437] {3}[Hardware Error]: 00000140: 00000012 80001001 00000000 00000001
................
[62274.888465] {3}[Hardware Error]: 00000150: 00000013 80001001 00000000 00000001
................
```

```
[62274.888490] {3}[Hardware Error]: 00000160: 00000014 80001001 00000000 00000001
................
[62274.888517] {3}[Hardware Error]: 00000170: 00000015 80001001 00000000 00000001
................
[62274.888543] {3}[Hardware Error]: 00000180: 00000016 80001001 00000000 00000001
................
[62274.888570] {3}[Hardware Error]: 00000190: 00000017 80001001 00000000 00000001
................
[62274.888596] {3}[Hardware Error]: 000001a0: 00000018 80001001 00000000 00000001
................
[62274.888625] {3}[Hardware Error]: 000001b0: 00000019 80001001 00000000 00000001
................
[62274.888652] {3}[Hardware Error]: 000001c0: 0000001a 80001001 00000000 00000001
................
[62274.888678] {3}[Hardware Error]: 000001d0: 0000001b 80001001 00000000 00000001
................
[62274.888704] {3}[Hardware Error]: 000001e0: 0000001c 80001001 00000000 00000001
................
[62274.888731] {3}[Hardware Error]: 000001f0: 0000001d 80001001 00000000 00000001
................
[62274.888756] {3}[Hardware Error]: 00000200: 0000001e 80001001 00000000 00000002
................
[62274.888782] {3}[Hardware Error]: 00000210: 0000001f 80001001 00000000 00000002
................
```

CPER Decoding

Common Platform Error Records (CPERs) is an error record format that is defined in Appendix N of the Unified Extensible Firmware Interface (UEFI) Specification v2.2, for reporting platform hardware errors. A CPER can be informational or can point to an error. Refer to *Unified Extensible Firmware Interface (UEFI) Specification Release 2.10* for more information about CPERs and the format.

Figure 7. UEFI-Defined CPER Format



NVIDIA-Defined CPER Format

NVIDIA defines its own section format. An NVIDIA-defined CPER can be identified by the CPER GUID. For all NVIDIA defined CPERs the CPER GUID is `6d5244f2-2712-11ec-bea7-`**`cb3fdb95c786`**. Table 19 lists the fields that make up an NVIDIA-defined section, their corresponding offset, and length.

Table 19. NVIDIA-Defined CPER Formats

Section	Byte Offset (in decimal)	Byte Offset (in hexadecimal)	Byte Length	Description
IP Signature	0	0	16	\0 terminated ASCII String.
Error Type	16	10	2	Unique by error type
Error Instance	18	12	2	Unique by error type
Severity	20	14	1	ACPI-defined severity: 0x00 = Correctable 0x01 = Fatal 0x02 = Corrected 0x03 = None
Socket Number	21	15	1	Socket number where the error occurred
Number of register data pairs	22	16	1	Number of offset/value pairs recorded in the block. Unique by error type.
Reserved	23	17	1	Always 0
Instance Base	24	18	8	Physical address of the instance. Can be used to clearly identify which instance caused the error and which socket.
Register Address/Value Pairs	32	20	2x8xN	Register address/value pairs (64 bit each)

Note The Register Address/Value pair pattern will repeat in certain cases.

Error Names

Table 20 provides a list of the error names.

Table 20. Error Names

Error Name	Description	GUID
C2C-IP-FAIL	OS-triggered FLR reset of GPU attempted. Interrupts triggered but RAS_FW unable to clear. Interrupts disabled. Other C2C and CLINK CPERs possible.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
CMET-FULL	Common Memory Error Table (CMET): Seen when total count of socket errors exceed threshold.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786

Error Name	Description	GUID
CMET-SHA256	Common Memory Error Table (CMET): Seen when error is reported in either primary or secondary SPI flash tables are invalid (corrupt, bad signature). System will recover to best possible state.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
DRAM-CHANNELS	Seen on every boot identifying the 30 best channels that are active for this boot.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
GPU-CONTNMT	State of GPU Containment process.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSM U/I ERROR	Hardware Safety Manager (HSM) error which couldn't be uniquely identified. Possibly fatal condition, error severity promoted to contain error condition.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSM_SW_ERR	Hardware Safety Manager (HSM) SW errors.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSS-IDLE	Informational error. Only observed when in error injection mode. It indicates that the HSS is disabled.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSS-BUSY	Informational error. Only observed when in error injection mode. Indicates HSS being enabled.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
L0/L1/L2 RESET	Level N reset source.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
C2C	An error has occurred in Chip-to-chip link	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
CCPLEXGIC	An error has occurred in the GIC.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
Clink	A Clink error has been detected.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
DCC-COH	An error has occurred in the device coherency cache.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
DCC-ECC	An ECC (error correction code) error has been caught in the device coherency cache.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSM	Hardware Safety Manager has detected an error.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
MCF	An error has occurred on a memory controller fabric.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786

Error Name	Description	GUID
PCIe	This event provides implementation specific register content for PCIe error events.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
SMMU	The SMMU hardware has encountered an error in cache and recovered.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
SOCHUB	A client without poison support received poisoned data	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
GPU-STATUS	Seen when GPU is not seen on PCIe controller, invalid configuration parameters are passed, or other C2C initialization issues are experienced.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
PAGES-RETIRED	Informational CPER. Only seen when in error injection mode. Will be sent when HSS idle is seen. Event indicating that pages have been retired.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
PCIE-DPC	PCIe downstream port containment has been triggered.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
CCPLEXSCF	An error has occurred in the Scalable Coherency Fabric	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
HSM_FABRIC	Hardware Safer Manager has detected a transaction timeout.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
FWERROR	This is a FW crash dump, so would have been the result of some FW issue.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
CMET-INFO	This is an information CPER containing CMET information. This CPER is only seen on performing certain EINJ commands.	6D5244F2-2712-11EC-BEA7-CB3FDB95C786
Platform Memory Generic	Corrupted memory has been detected and poisoned.	61EC04FC-48E6-D813-25C9-8DAA44750B12
Processor Generic	An error has been handled on an application processor	E19E3D16-BC11-11E4-9CAA-C2051D5D46B0
PCIe Generic	A PCIe device has reported an error through AER standard error reporting.	D995E954-BBC1-430F-AD91-B44DCB3C6F35

DRAM Error Reporting Using CPERs

This section provides information about reporting errors in the DRAM using CPERs.

FWERROR

Table 22 provides the format of the CPERs that are generated by MB1 and sent to BMC through Reliability, Availability, and Serviceability (RAS). These CPERs are now enhanced to include additional information about the state of the DRAM channels.

Table 21. FWERROR Data Path and Size

Component	Value
Data Path	MB1 → RAS → SatMC → BMC
Size	672 Bytes

Table 22. FWERROR Format

Field	Byte Offset	Byte Length	Value
IP Signature	0	16	"FWERROR"
Error Type	16	2	TEGRABL_NV_CPER_TYPE_DEFAULT (0x0000)
Error Instance	18	2	0x0000
Severity	20	1	EFI_GENERIC_ERROR_FATAL (0x01)
Socket Number	21	1	0x00 - 0x03
Number of register values	22	1	0x50 (SCRATCH_21 to SCRATCH_100)
Reserved	23	1	0x0
Instance Base	24	8	NV_ADDRESS_MAP_SCRATCH_BASE + SCRATCH_SCRATCH_21
Value0 (SCRATCH_SCRATCH_21)	+8	8	Firmware component initiating the FWERROR ("MB1")
Value1 (SCRATCH_SCRATCH_22)	+8	8	Task checkpoint
Value2 (SCRATCH_SCRATCH_23)	+8	8	MB1 Error Code
Value 3 - Value 18 (SCRATCH_SCRATCH_24 - SCRATCH_SCRATCH_39)	+8	8	"MB1 version string"

Field	Byte Offset	Byte Length	Value
Value 19 (SCRATCH_SCRATCH_40)	+8	8	Mask of channels retired due to the number of bad pages > Threshold Threshold = Mem size per channel * Channel max bad pages per GB Channel max bad pages per GB defaults to 4 and is only configurable for SKU_893, through the MB1-BCT (nv_int_config_1) and the UEFI Menu.
Value 20 (SCRATCH_SCRATCH_41)	+8	8	Mask of channels retired due to training or alias check failures
Value 21 (SCRATCH_SCRATCH_42)	+8	8	Channel with the worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 22 (SCRATCH_SCRATCH_43)	+8	8	Channel with second worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 23 (SCRATCH_SCRATCH_44)	+8	8	Channel with third worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 24 (SCRATCH_SCRATCH_45)	+8	8	Bitmap of channels that failed the boot frequency training or the alias checker
Value 25 (SCRATCH_SCRATCH_46)	+8	8	Value of the SCRATCH_DRAM_CHANNEL_DISABLE register: Bits[0:0] SIG_1 Bits[1:5] CHANNEL_1 Bits[6:6] RETRIED_1 Bits[7:7] SIG_2 Bits[12:8] CHANNEL_2 Bits[13:13] RETRIED_2 Bits[14:14] OVERFLOW
Value 26 - Value 79 (SCRATCH_SCRATCH_47-SCRATCH_SCRATCH_100)	+8	8	Reserved

FWERROR1

These new CPERs are generated by MB1 and sent directly to BMC over SSIF and contain the same information as the FWERROR CPER, but FWERROR1 only includes information about scratch registers 21-60 and are reported as 32-bit values instead of as 64-bit values.

Note NVIDIA's reference BMC does not yet support the reception of these CPERs.

Table 23. FWERROR1 Data Path and Size

Component	Value
Data Path	MB1 → BMC
Size	192 Bytes

Table 24. FWERROR1 Format

Field	Byte Offset	Byte Length	Value
IP Signature	0	16	"FWERROR1"
Error Type	16	2	TEGRABL_NV_CPER_TYPE_DEFAULT (0x0000)
Error Instance	18	2	0x0000
Severity	20	1	EFI_GENERIC_ERROR_FATAL (0x01)
Socket Number	21	1	0x00 - 0x03
Number of register values	22	1	0x28 (SCRATCH_21 to SCRATCH_60)
Reserved	23	1	0x0
Instance Base	24	8	NV_ADDRESS_MAP_SCRATCH_BASE + SCRATCH_SCRATCH_21
Value0 (SCRATCH_SCRATCH_21)	+4	4	Firmware component initiating the FWERROR ("MB1")
Value1 (SCRATCH_SCRATCH_22)	+4	4	Task checkpoint
Value2 (SCRATCH_SCRATCH_23)	+4	4	MB1 Error Code
Value 3 - Value 18 (SCRATCH_SCRATCH_24 - SCRATCH_SCRATCH_39)	+4	4	"MB1 version string"
Value 19 (SCRATCH_SCRATCH_40)	+4	4	Mask of channels retired due to number of bad pages > threshold

Field	Byte Offset	Byte Length	Value
Value 20 (SCRATCH_SCRATCH_41)	+4	4	Mask of channels retired due to training or alias check failures
Value 21 (SCRATCH_SCRATCH_42)	+4	4	Channel with the worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 22 (SCRATCH_SCRATCH_43)	+4	4	Channel with 2nd worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 23 (SCRATCH_SCRATCH_44)	+4	4	Channel with 3rd worst number of bad pages. Bits[0:4]: Channel number Bits[5:31]: Number of bad pages
Value 24 (SCRATCH_SCRATCH_45)	+4	4	Bitmap of channels that failed boot frequency training or alias checker.
Value 25 (SCRATCH_SCRATCH_46)	+4	4	Value of SCRATCH_DRAM_CHANNEL_DISABLE Bits[0:0] SIG_1 Bits[1:5] CHANNEL_1 Bits[6:6] RETRIED_1 Bits[7:7] SIG_2 Bits[12:8] CHANNEL_2 Bits[13:13] RETRIED_2 Bits[14:14] OVERFLOW
Value 26 – Value 39 (SCRATCH_SCRATCH_47 – SCRATCH_SCRATCH_60)	+4	4	Reserved

FWERROR2

These new CPERs are generated by MB1 and sent directly to BMC over SSIF and contain the same information as the FWERROR CPER, but FWERROR2 only includes information about scratch registers 61-100 and are reported as 32-bit values instead of as 64-bit values.

Note NVIDIA's reference BMC does not yet support the reception of these CPERs.

Table 25. FWERROR2 Data Path and Size

Component	Value
Data Path	MB1 → BMC
Size	192 Bytes

Table 26. FWERROR2 Format

Field	Byte Offset	Byte Length	Value
IP Signature	0	16	"FWERROR2"
Error Type	16	2	TEGRABL_NV_CPER_TYPE_DEFAULT (0x0000)
Error Instance	18	2	0x0000
Severity	20	1	EFI_GENERIC_ERROR_FATAL (0x01)
Socket Number	21	1	0x00 - 0x03
Number of register values	22	1	0x28 (SCRATCH_61 to SCRATCH_100)
Reserved	23	1	0x0
Instance Base	24	8	NV_ADDRESS_MAP_SCRATCH_BASE + SCRATCH_SCRATCH_61
Value 0 – Value 39 (SCRATCH_SCRATCH_61 – SCRATCH_SCRATCH_100)	+4	4	Reserved

CMET-INFO

These existing CPERs are generated by RAS, in response to EINJ commands, and sent to BMC through SatMC. These CPERs are now enhanced to include the reason for channel disablement.

Table 27. CMET-INFO Data Path and Size

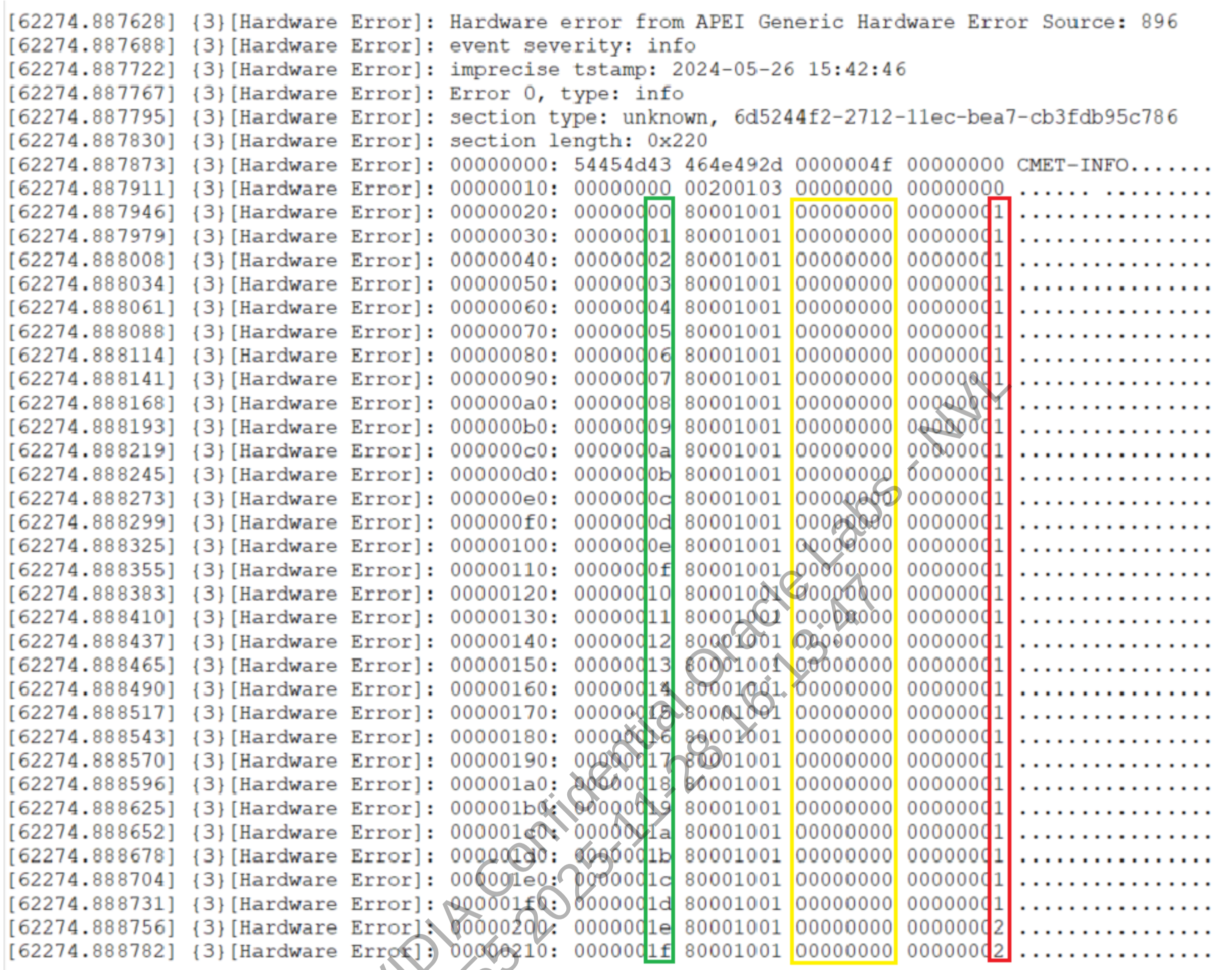
Component	Value
Data Path	RAS → SatMC → BMC
Size	544 Bytes

Table 28. CMET-INFO Format

Field	Byte Offset	Byte Length	Value
IP Signature	0	16	"CMET-INFO"
Error Type	16	2	0
Error Instance	18	2	0
Severity	20	1	Informational
Socket Number	21	1	0-3
Number of register data pairs	22	1	32
Reserved	23	1	0
Instance Base	24	8	0
Address	+8	8	Channel 0 Address
Value	+8	8	Here is the Channel 0-related information: Bits[0:31]: Error count Bit32: if set, channel is enabled Bit33: if set, channel is considered a spare (and is currently disabled) Bit34: if set, channel is permanently disabled Bit48-49: if channel is disabled, indicate the reason it was disabled: 00: alias checker failed 01: training at POR frequency failed 10: training at boot frequency failed 11: threshold of bad pages exceeded

Note The address and value fields repeat for channels 1-31.

Here is an example of a decoded CMET-INFO CPER.



```
[62274.887628] {3}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 896
[62274.887688] {3}[Hardware Error]: event severity: info
[62274.887722] {3}[Hardware Error]: imprecise tstamp: 2024-05-26 15:42:46
[62274.887767] {3}[Hardware Error]: Error 0, type: info
[62274.887795] {3}[Hardware Error]: section type: unknown, 6d5244f2-2712-11ec-bea7-cb3fdb95c786
[62274.887830] {3}[Hardware Error]: section length: 0x220
[62274.887873] {3}[Hardware Error]: 00000000: 54454d43 464e492d 0000004f 00000000 CMET-INFO.......
[62274.887911] {3}[Hardware Error]: 00000010: 00000000 00200103 00000000 00000000 ...... .........
[62274.887946] {3}[Hardware Error]: 00000020: 00000000 80001001 00000000 00000001 ................
[62274.887979] {3}[Hardware Error]: 00000030: 00000001 80001001 00000000 00000001 ................
[62274.888008] {3}[Hardware Error]: 00000040: 00000002 80001001 00000000 00000001 ................
[62274.888034] {3}[Hardware Error]: 00000050: 00000003 80001001 00000000 00000001 ................
[62274.888061] {3}[Hardware Error]: 00000060: 00000004 80001001 00000000 00000001 ................
[62274.888088] {3}[Hardware Error]: 00000070: 00000005 80001001 00000000 00000001 ................
[62274.888114] {3}[Hardware Error]: 00000080: 00000006 80001001 00000000 00000001 ................
[62274.888141] {3}[Hardware Error]: 00000090: 00000007 80001001 00000000 00000001 ................
[62274.888168] {3}[Hardware Error]: 000000a0: 00000008 80001001 00000000 00000001 ................
[62274.888193] {3}[Hardware Error]: 000000b0: 00000009 80001001 00000000 00000001 ................
[62274.888219] {3}[Hardware Error]: 000000c0: 0000000a 80001001 00000000 00000001 ................
[62274.888245] {3}[Hardware Error]: 000000d0: 0000000b 80001001 00000000 00000001 ................
[62274.888273] {3}[Hardware Error]: 000000e0: 0000000c 80001001 00000000 00000001 ................
[62274.888299] {3}[Hardware Error]: 000000f0: 0000000d 80001001 00000000 00000001 ................
[62274.888325] {3}[Hardware Error]: 00000100: 0000000e 80001001 00000000 00000001 ................
[62274.888355] {3}[Hardware Error]: 00000110: 0000000f 80001001 00000000 00000001 ................
[62274.888383] {3}[Hardware Error]: 00000120: 00000010 80001001 00000000 00000001 ................
[62274.888410] {3}[Hardware Error]: 00000130: 00000011 80001001 00000000 00000001 ................
[62274.888437] {3}[Hardware Error]: 00000140: 00000012 80001001 00000000 00000001 ................
[62274.888465] {3}[Hardware Error]: 00000150: 00000013 80001001 00000000 00000001 ................
[62274.888490] {3}[Hardware Error]: 00000160: 00000014 80001001 00000000 00000001 ................
[62274.888517] {3}[Hardware Error]: 00000170: 00000015 80001001 00000000 00000001 ................
[62274.888543] {3}[Hardware Error]: 00000180: 00000016 80001001 00000000 00000001 ................
[62274.888570] {3}[Hardware Error]: 00000190: 00000017 80001001 00000000 00000001 ................
[62274.888596] {3}[Hardware Error]: 000001a0: 00000018 80001001 00000000 00000001 ................
[62274.888625] {3}[Hardware Error]: 000001b0: 00000019 80001001 00000000 00000001 ................
[62274.888652] {3}[Hardware Error]: 000001c0: 0000001a 80001001 00000000 00000001 ................
[62274.888678] {3}[Hardware Error]: 000001d0: 0000001b 80001001 00000000 00000001 ................
[62274.888704] {3}[Hardware Error]: 000001e0: 0000001c 80001001 00000000 00000001 ................
[62274.888731] {3}[Hardware Error]: 000001f0: 0000001d 80001001 00000000 00000001 ................
[62274.888756] {3}[Hardware Error]: 00000200: 0000001e 80001001 00000000 00000002 ................
[62274.888782] {3}[Hardware Error]: 00000210: 0000001f 80001001 00000000 00000002 ................
```

> The channel number can be identified by the characters in the green box.
> The error count for each channel can be identified by the characters in the yellow box.
> A channel can be ENABLED, SPARED, or DISABLED.

The channel status is highlighted by the red box, and its value can be one of the following:

- Enabled = 1
- Spare = 2
- Permanently disabled = 4

> The total number of available channels can be identified by counting all Enabled (1) channels.
> The total number of spare channels can be identified by counting all Spare (2) channels.

Decoded CPER Examples

This section provides decoded CPER examples.

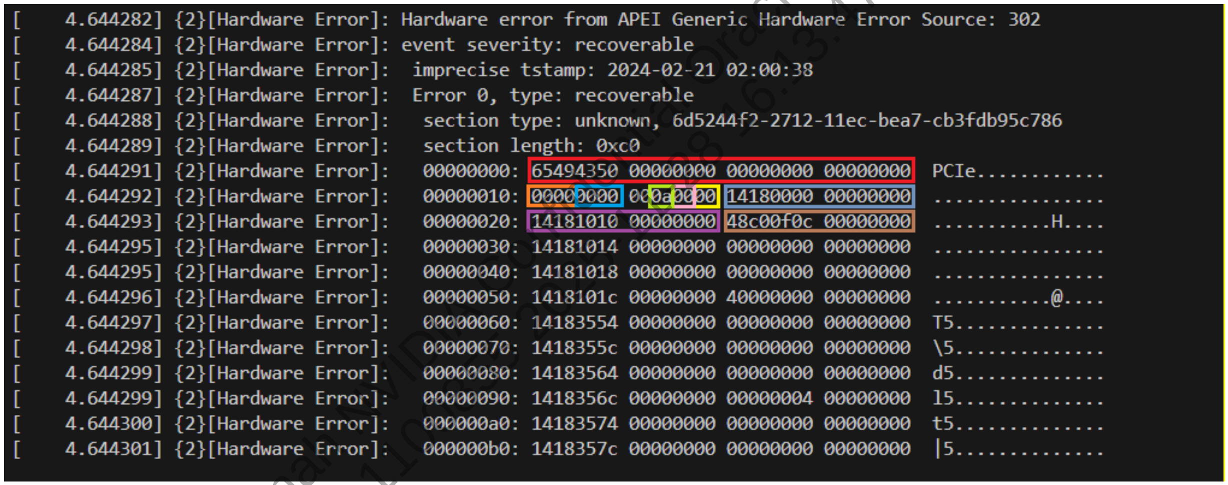
Note:

- The decoded CPER fields can be cross-referenced in the *NVIDIA Grace CPER Catalog* (NVOnline: **1115302**).
- The dmesg CPER log will follow little endian format.

PCIe Example

Figure 8 is a PCIe example.

Figure 8. A PCIe Example



```
[    4.644282] {2}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 302
[    4.644284] {2}[Hardware Error]: event severity: recoverable
[    4.644285] {2}[Hardware Error]:  imprecise tstamp: 2024-02-21 02:00:38
[    4.644287] {2}[Hardware Error]:  Error 0, type: recoverable
[    4.644288] {2}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-cb3fdb95c786
[    4.644289] {2}[Hardware Error]:   section length: 0xc0
[    4.644291] {2}[Hardware Error]:   00000000: 65494350 00000000 00000000 00000000  PCIe............
[    4.644292] {2}[Hardware Error]:   00000010: 00000000 000a0000 14180000 00000000  ................
[    4.644293] {2}[Hardware Error]:   00000020: 14181010 00000000 48c00f0c 00000000  ...........H....
[    4.644295] {2}[Hardware Error]:   00000030: 14181014 00000000 00000000 00000000  ................
[    4.644295] {2}[Hardware Error]:   00000040: 14181018 00000000 00000000 00000000  ................
[    4.644296] {2}[Hardware Error]:   00000050: 1418101c 00000000 40000000 00000000  ...........@....
[    4.644297] {2}[Hardware Error]:   00000060: 14183554 00000000 00000000 00000000  T5..............
[    4.644298] {2}[Hardware Error]:   00000070: 1418355c 00000000 00000000 00000000  \5..............
[    4.644299] {2}[Hardware Error]:   00000080: 14183564 00000000 00000000 00000000  d5..............
[    4.644299] {2}[Hardware Error]:   00000090: 1418356c 00000000 00000004 00000000  l5..............
[    4.644300] {2}[Hardware Error]:   000000a0: 14183574 00000000 00000000 00000000  t5..............
[    4.644301] {2}[Hardware Error]:   000000b0: 1418357c 00000000 00000000 00000000  |5..............
```

Decoded CPER Fields

Here are the decoded CPER fields:

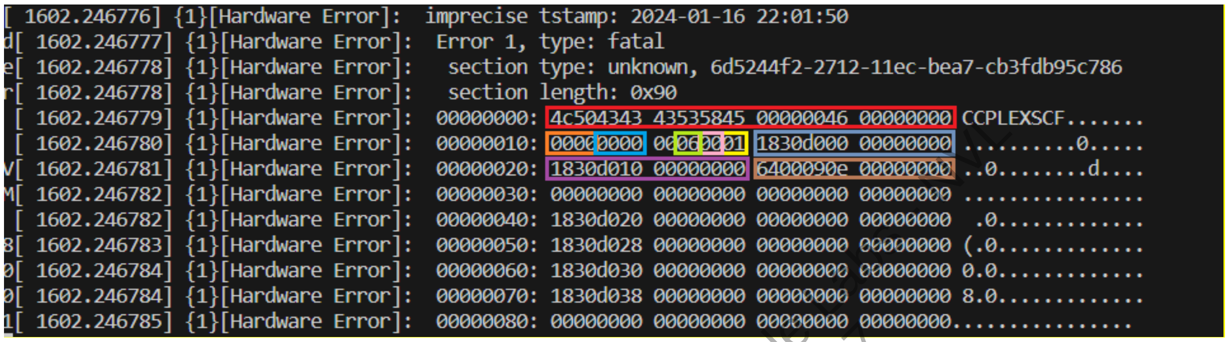
- IP Signature: PCIe
- Error Type: 0x0000
- Error Instance: 0x0000
- Severity: 0x00 = Correctable
- Socket Number: 0x00
- Number of register data pairs: 0a
- Instance Base: 00000000 14180000
- Register Address/Value Pairs:
- Address of register at offset 0x20: 00000000 14181010

> Value of register at offset0x20: 00000000 48c00f0c

A CCPLEXSCF Example

Figure 9 shows a CCPLEXSCF example.

Figure 9. A CCPLEXSCF Example



```
[ 1602.246776] {1}[Hardware Error]:  imprecise tstamp: 2024-01-16 22:01:50
d[ 1602.246777] {1}[Hardware Error]:  Error 1, type: fatal
e[ 1602.246778] {1}[Hardware Error]:   section type: unknown, 6d5244f2-2712-11ec-bea7-cb3fdb95c786
r[ 1602.246778] {1}[Hardware Error]:   section length: 0x90
 [ 1602.246779] {1}[Hardware Error]:  00000000: 4c504343 43535845 00000046 00000000 CCPLEXSCF.......
 [ 1602.246780] {1}[Hardware Error]:  00000010: 00000000 00060001 1830d000 00000000 ..........0.....
V[ 1602.246781] {1}[Hardware Error]:  00000020: 1830d010 00000000 6400090e 00000000 ..0........d....
M[ 1602.246782] {1}[Hardware Error]:  00000030: 00000000 00000000 00000000 00000000 ................
 [ 1602.246782] {1}[Hardware Error]:  00000040: 1830d020 00000000 00000000 00000000  .0.............
8[ 1602.246783] {1}[Hardware Error]:  00000050: 1830d028 00000000 00000000 00000000 (.0.............
0[ 1602.246784] {1}[Hardware Error]:  00000060: 1830d030 00000000 00000000 00000000 0.0.............
0[ 1602.246784] {1}[Hardware Error]:  00000070: 1830d038 00000000 00000000 00000000 8.0.............
1[ 1602.246785] {1}[Hardware Error]:  00000080: 00000000 00000000 00000000 00000000................
```

Decoded CPER Fields

Here are the decoded CPER fields:

> IP Signature: CCPLEXSCF
> Error Type: 0x0000
> Error Instance: 0x0000
> Severity: 0x01 = Fatal
> Socket Number: 0x00
> Number of register data pairs: 06
> Instance Base: 00000000 1830d000
> Register Address/Value Pairs:
> Address of register at offset 0x20: 00000000 1830d010
> Value of register at offset 0x20: 00000000 6400090e

CPER Severities

Table 29 provides the possible ACPI severity that can be observed for different types of CPERs.

Table 29. ACPI Severity Mappings

IP Block	NVIDIA Specific	ACPI Error Severity Mapping
PCIe	Y	NONE
PCIe-DPC	Y	CORRECTED

IP Block	NVIDIA Specific	ACPI Error Severity Mapping
PCIe	N	RECOVERABLE
PCIe	N	CORRECTED
DRAM	N	RECOVERABLE
DRAM	N	FATAL
DRAM	Y	NONE
DRAM	Y	CORRECTED
CCPLEX SCF	Y	CORRECTED
CCPLEX SCF	Y	FATAL
CCPLEX GIC	Y	FATAL
GGPLEX GIC	Y	CORRECTED
CCPLEX	N	FATAL
CCPLEX	N	CORRECTED
MCF	Y	RECOVERABLE
C2C	Y	CORRECTED
C2C	Y	FATAL
L2 reset	Y	FATAL
SOCHub	Y	FATAL
SMMU	Y	CORRECTED
CPUEA	N	NONE
HSS	Y	NONE
HSM	Y	RECOVERABLE
HSM	Y	FATAL
HSM	Y	CORRECTED
Clink	Y	FATAL
Clink	Y	CORRECTED
FWERROR	Y	FATAL
FWERROR	Y	CORRECTED
DCC-ECC	Y	FATAL
DCC-ECC	Y	CORRECTED
DCC-COH	Y	FATAL
CMET	Y	CORRECTED
CMET	Y	NONE
GPU	Y	RECOVERABLE
GPU	Y	NONE

Summary

Grace provides extensive RAS capabilities that comply with Arm RAS v8.4 such as the following:

- The SEC/DED ECC is deployed across the design (busses, caches, and RAMs) and on internal interfaces such as LPDDR5x.
- Memory features including scrubbing and page off lining are used to ensure memory integrity.
- Serial interfaces provide CRC protection on external and ECC protection on internal RAMs.
- PCIe Gen5 root ports support Downstream Port Containment and Advanced Error Reporting.

Internal features such as lockstep, bus integrity, timeouts (including watchdog), voltage, and temperature monitors provide additional robustness to the design. Arm standards are used to determine how errors are handled and reported to the system.

References and Bibliography

ARM. (2018). SBSA ECR: Error handling requirements for Root ports, host bridges, Root Complex Integrated Endpoints and Integrated Endpoints behind a root port. ARM.

ARM. (2019, Jul 1). *Arm Reliability, Availability, and Serviceability (RAS) Specification Armv8, for the Armv8-A architecture profile.* Retrieved from ARM Developer Website: https://developer.arm.com/documentation/ddi0587/latest

Meza, J., Wu, Q., Kumar, S., & Mutlu, O. (2015). Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* Rio de Janeiro.