An Overview of RAS for Compute Express Link® Covering from CXL® 2.0 to CXL® 3.1

Antonio Hasbun, Intel
Jordan Chin, Dell
Daniele Balluchi, Micron
Thomas Won Ha Choi, SK hynix
Tony Benavides, Intel
Sanjay Goyal, Rambus

March 27, 2024

1 Contents

2 Introduction and Motivation

During the evolution of CXL from its released version 2.0 to the version 3.1 released in 2023 there have been a lot of improvements in the standardization of Reliability, Availability and Serviceability (RAS) control and reporting. The objective of this whitepaper is to provide clarity for the reasons behind and use cases of these features.

As we expand coherent memory into the disaggregated architectures of the future, RAS continues to be top of mind. One of the fundamental limitations to the future architectures will be the reliability of the system; and therefore, the CXL Consortium has dedicated efforts towards standardization of the RAS qualities that can be required for these systems.

This whitepaper is organized in the following way. The next chapter details all the new functionalities that have been defined for RAS since CXL 2.0. This includes “features” and some that can trigger RAS actions using the maintenance commands. After that, the next section details some of the improvements made to the specification that provide greater clarity to RAS actions and reporting. The fourth chapter provides examples of flows and recipes of how the RAS actions are envisioned to take place in the system. That chapter provides the real-life examples of the usage models for some of the most debated RAS actions in the CXL specification. Finally, there is a chapter with conclusion that summarizes the key takeaways from this whitepaper.

The content in this whitepaper is not to be considered normative. The CXL 3.1 specification remains the sole source of CXL RAS capabilities and requirements.

For more details about the CXL standard, key concepts, and CXL terminology, please visit the CXL Consortium website (https://www.computeexpresslink.org/) for whitepapers and specification documents.

3 New RAS Features in CXL 3.1

Some of the important changes that were introduced in CXL 3.1 are the features and the maintenance command. The features refer to "configuration, control or capability[1]" that are being exposed to the management agent. Some RAS capabilities are exposed through features such that they can be configured through attributes or triggered as a capability through the maintenance command.

Not all features are required to support a maintenance command; just the features that can trigger a RAS action have that capability. In many cases, the features will control how a RAS feature is configured and will also control how the result is reported. Some features will create event logs when the actions are completed, while others update the corresponding system logs.

It is important to highlight some common functionalities in the features systems; for they apply to all features and support some RAS use models.

The implementation of features for RAS is optional and as such the features are discovered in a programmatical way. The "Get Supported Features" command will enable the management agent to list the features that are available for that particular hardware. Features are enumerated in the specification and recognized by the UUID that is unique to each one. This provides the greatest range of flexibility while allowing for enough standardization to make useful generic drivers.

The features specified are not required for CXL devices, but the specification provides the basis so that adding this high-value RAS functionalities is simple and standard across CXL devices.

Features in general have characteristics that describe their behavior with respect to reset and how the configuration can be managed.

Features can have up to three different sets of values for their attributes: the default value; the saved value and the current value. The default values are the ones that are programmed into the device by the manufacturer. The "Saved Values"; if supported, are values that can be saved into the device for use after certain resets (for details see table 8-91 "Feature Attribute(s) Values after Reset"). Finally, the current values are the ones that have been set after reset and are currently acting on the device. The attributes from the different sets can be read using the Get Feature Command, while the Set Feature can write the current and saved values if available.

Maintenance operation features have Operation Capabilities and Operation Modes that help determine if the feature should be device-initiated or host-initiated. The Operation Capabilities advertises if the device can trigger its own feature, while the Operation Mode is the actual behavior that is being used. The host can change this Operation mode with a Set Feature command if the Operation Capability bit was set.

All features support versioning, and they are designed to be backwards compatible. The versions of a feature are read through the "Get Supported Features" command; where each feature will have a version for its "Get Feature" and another for its "Set Feature". The way the backwards compatibility works is as follows:

[1] CXL specification 3.0 Chapter 8.2.8.6

- If a feature definition has changed (due to a new revision on the specification) and returns a new output payload for the “Get Feature” command; then it should increment the “Get Feature Version”; however, all the new attributes and extra data that is supplied in the new revision can be safely ignored by older software and the feature should work just as it did for previous versions. If a previous attribute is no longer relevant for the new version of the feature; then its contents must be fixed to a safe value (that is returned with the “Get Feature” command) so that the previous functionality is preserved.
- If the feature has changed the input of “Set Feature” command; then it should increment the “Set Feature Version.” In order to preserve the backwards compatibility, the size of the Set Feature payload can only increase; the new attributes must be optional, and the feature must behave as intended in previous version when those attributes are not provided. On the other hand, if some attributes for the feature are no longer needed, they can be ignored by the device if the previous functionality is preserved.

Some features have RAS actions that can be initiated; those are triggered through a “Perform Maintenance” command. The “Perform Maintenance” is a command that runs very specific actions depending on the Operation Class and Sub Class that is being selected. The maintenance operations can have several restrictions and run in the foreground or in the background; it all depends in the specific class and subclass that is being targeted.

The following section will dive into each feature that was added and we will show its intended use and the rationale behind them.

3.1 New Component ID and its Uses

The CXL 3.1 specification introduces several changes aimed at providing a uniform and consistent method for associating event records to a CXL memory device’s components. Two of these changes are particularly noteworthy:

1. The Component Identifier field has been standardized using industry standard management entity semantics. Specifically, the DMTF PLDM Monitoring & Control and DMTF Redfish Device Enablement (RDE) semantics. This helps ensure that Component Identifier is consistently defined across different devices and management entities, facilitating efficient and effective identification of issues.

2. The Component Identifier field has been newly added to several event records and logs, further increasing component identification capabilities in a wider variety of situations and issues.

In this section of the CXL 3.1 RAS whitepaper, we will examine these changes in more detail, as well as discuss how devices and management entities should properly utilize and make the most of the updated Component Identifier field.

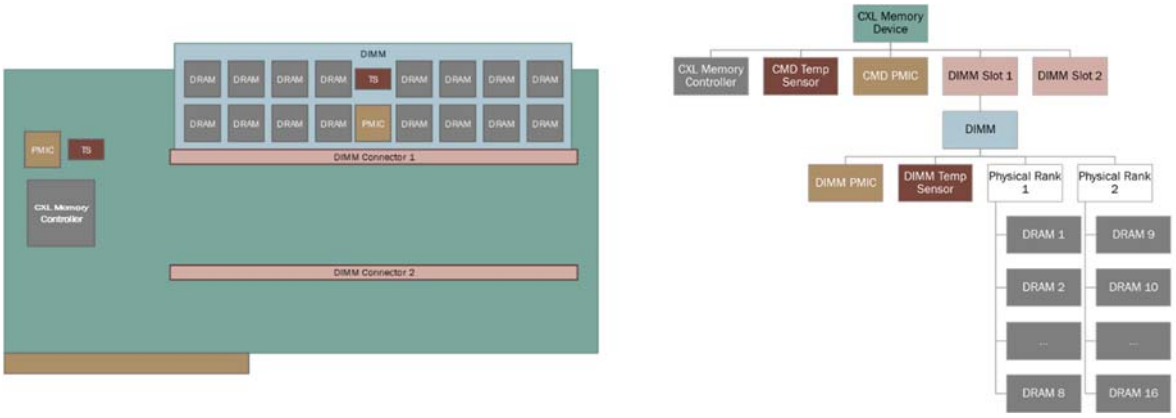
Today the CXL specification provides a Component Identifier field as a means to associate a component to an event record. However, prior to the CXL 3.1 specification, the formatting of the Component Identifier was device specific, and its use was limited to just General Media Event Records. The term ‘component’ is intentionally generic in the CXL specification and enables using the Component Identifier field to identify any type of component. In some situations, it may refer to individual components such as a single DRAM chip, CXL controller, power management ICs, or other logic device. In other situations, it may refer to an

entire field replaceable unit (FRU) such as a DIMM or battery backup unit. Due to the device-specific nature of the Component Identifier format and definition, it was not possible to determine the type and relationship of the component the field pointed to without prior knowledge.

Therefore, in addition to device-specific formatting, the CXL 3.1 specification implements a standardized formatting of the Component Identifier field based on device agnostic DMTF management semantics:

- PLDM Entity Identification Information used by DMTF PLDM Monitoring & Control
  - Defined in Entity Identification Information section 9.1 of the DMTF Platform Level Data Model for Platform Monitoring and Control Specification (DSP0248 version 1.2.2)
- Resource ID used by DMTF Redfish Device Enablement (RDE)
  - Defined in Redfish Resource PDR section 28.26 of the DMTF Platform Level Data Model for Platform Monitoring and Control Specification (DSP0248 version 1.2.2)

These management semantics are scalable and allow the device to convey hierarchical component/entity relationships to managing entities in a manner as shown by the figure below. Following the example figure further – by pointing to DRAM 2 in the Component Identifier field, managing entities will be able to ascertain that DRAM 2 is part of the logical entity group 'Physical Rank 1', which is physically part of a DIMM connected to DIMM Slot 1, and subsequently a sub-component of the larger CXL Memory Device.



Use of either identification schemes are optional; CXL Memory Devices may continue to use device-specific formatting if desired. However, if either are used, it is imperative that component relationships and associations are appropriately established and conveyed by the CXL Memory Device in their respective schemas. Further details on how to do this is out of scope for this RAS whitepaper but PLDM guidance can be found in the CXL Memory Device PLDM Model published by the CXL Consortium.

Note: Component enumeration and identification down to the DRAM component level is recommended but not required. In fact, the CXL Memory Device PLDM Model itself is only informational. Throughout the remainder of this section, it is stated that the Component Identifier field should point to the lowest identifiable component(s) associated with the event. At a minimum it is highly recommended that critical logic components (e.g., PMICs) and FRU components (e.g., DIMMs) are enumerated and identified.

To use either format, the CXL Memory Device must:

1. Set the Component Identifier field valid flag in the Validity Flags field of the applicable payload.

2. Set the Component Identifier governance flag in the Validity Flags field of the applicable payload.
3. Set either (or both) PLDM Entity ID or Resource ID flags in the first byte of the Component Identification field.
4. Populate the remainder of the field using the definition provided in Table 8-43 of the CXL 3.1 specification.

Another significant change in CXL 3.1 is the addition of the Component Identifier Field to the DRAM Event Records, Memory Module Event records, and Memory Sparing Event records. It has also been added to other device output payloads such as the DDR5 Error Check Scrub Log and Media Test Results Log. Guidance on how Component Identifier is used with the event records and logged is provided:

- General Media and DRAM Event Records – The Component Identifier field should point to the lowest identifiable component(s) associated with the event. In hierarchical terms, the CXL Memory Device should first try to identify the DRAM or memory media component(s) as the most granular identifiable component. If the individual media component(s) cannot be ascertained, then the CXL Memory Device should try to identify the physical/logical rank(s) or group(s) of media components associated with the event. If this cannot be ascertained, then the CXL Memory Device should try to identify the FRU associated. If multiple components are identified in a single event, then multiple event records should be generated for each component involved. The CXL 3.1 specification provides an 'Implementation Note' for a few common memory error/failure scenarios that are repeated here:
    - An uncorrectable error occurs and the CXL Memory Device is able to isolate to a few specific DRAM components. The CXL Memory Device should generate several event records for the uncorrectable error, each pointing to the identified DRAM components in the Component Identifier field.
    - An uncorrectable error occurs and the CXL Memory Device is not able to isolate to any specific DRAM components. The CXL Memory Device should generate one event record for the uncorrectable error, pointing to the identified rank of DRAM components in the Component Identifier field.
    - A DDR bus training failure occurs in a channel populated with two DIMMs. The CXL Memory Device is unable to isolate which DRAM component, rank, or DIMM is failing. The CXL Memory Device should generate two event records for the channel training failure, each pointing to one of the identified DIMMs in the Component Identifier field.
- Memory Module Event Records – Since this event record is intended to log events related to components other than memory media, the Component Identifier should point directly to the identifiable logic component(s) associated with the event (e.g., PMICs, SPDs, etc.). If the individual logic component(s) cannot be ascertained, then the CXL Memory Device should try to identify the FRU associated. A few common module error/failure scenarios are provided as examples:
    - An unsupported DIMM has been installed. The CXL Memory Device should generate one Memory Module Event Record, pointing to the unsupported DIMM in the Component Identifier field.
    - A power fault occurs on an installed DIMM with a power management IC (PMIC). The CXL Memory Device should generate one Memory Module Event Record, pointing to the DIMM's PMIC in the Component Identifier field.

- A logic component on a DIMM has hung the 2-wire interface used for sideband communication with two DIMMs on a channel. Since the CXL Memory Device cannot ascertain which logic component between the two DIMMs has hung the bus, the CXL Memory Device should generate two Memory Module Event Records, pointing to both DIMMs in the Component Identifier field.

- DDR5 Error Check Scrub Log and Media Test Results Log – Like DRAM Event Records, these two logs should point to the specific DRAM component being identified by the scrub or media test results. See the DDR5 Error Check Scrub and Media Test sections in this RAS whitepaper for further details on these features.

In summary, the CXL 3.1 specification brings significant changes to the Component Identifier field that aims to provide uniformity and consistency in associating event records to CXL Memory Devices' components. With this update, the field has been added to several event records and logs, including DRAM Event Records, Memory Module Event records, and Memory Sparing Event records, among others. Additionally, the specification provides guidance on how to use the Component Identifier field with these event records and logs. The Component Identifier field has also been standardized using device agnostic and industry standard management entity semantics, specifically in DMTF PLDM Monitoring & Control and DMTF Redfish Device Enablement semantics. This standardization will help ensure consistent Component Identifier definitions across different devices and management entities.

3.2 Advanced CVME Threshold

Prior to CXL 3.1, the Set Alert Configuration CCI command was the only method available for management entities to threshold corrected volatile memory errors (CVMEs) on CXL Memory Devices. This basic method was limited in a number of ways:

1. The programmable CVME threshold was a single value applied to the entire range of Host-managed Device Memory (HDM), regardless of memory capacity or memory component topology.
2. The programmable CVME threshold could only generate a warning-severity event record.
3. Once the CVME threshold had been exceeded and warning event record generated, the CXL memory device could not generate subsequent event records without being reprogrammed.

The CXL 3.1 specification introduces an optional Advanced Programmable Corrected Volatile Memory Error (CVME) Threshold feature that addresses the above limitations and adds further capabilities:

- Programmable thresholds to generate a memory media event record at Informational, Warning, and/or Failure severity levels.
- Configurable whether to use the ‘HW Replacement Needed’ flag for each event record severity.
- Configurable error counter granularity for thresholds (e.g., per memory media FRU or per rank counters).
- Configurable mask for counting single-bit or corrected multi-bit errors towards thresholds.
- Programmable thresholds for errors detected during patrol scrub operations.
- Configurable expiration and automatic reset of CVME threshold counters after some period of programmable time.

The command set leverages the Get and Set Features interface to discover the device capabilities and set the configurations of the advanced programmable corrected volatile memory error threshold feature.

This feature cannot be used simultaneously with the corrected volatile memory error programmable threshold via the Set Alert Configuration or internal CVMEs thresholds pre-programmed within the device. Therefore, when this interface is used to program and enable advanced CVME thresholds, any corrected volatile memory error threshold programmed via the Set Alert Configuration or any internal pre-programmed CVME thresholds shall be automatically disabled.

3.2.1 Supplemental Details of the Advanced Programmable CVME Threshold Feature

The following is provided to supplement the individual attribute information in the Advanced Programmable CVME Threshold feature of the CXL 3.1 specification.

Corrected Volatile Memory Error (CVME) Threshold Granularity – This attribute determines whether the programmed threshold applies to all host-managed device memory (HDM) on the CXL device, per memory media FRU (e.g., DIMM), or per rank. It is not possible to set different thresholds for different DIMMs or ranks. CXL memory devices that do not support memory media FRUs would not support the respective granularity option.

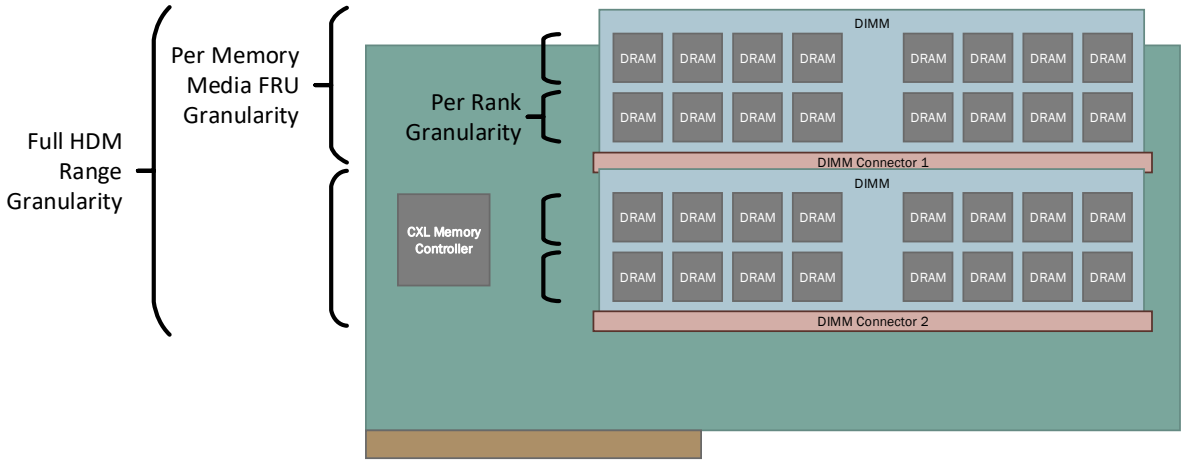


Figure 1 - Graphical Illustration of Granularity Options

Example: For a CXL memory device with two DIMMs where granularity is set to 'per memory media FRU' and programmed warning threshold is set to 100. The CXL memory device tracks CVMEs separately for each DIMM. When each DIMM CVME count exceeds the programmed warning threshold, the device will generate separate General/DRAM warning events for each excursion.

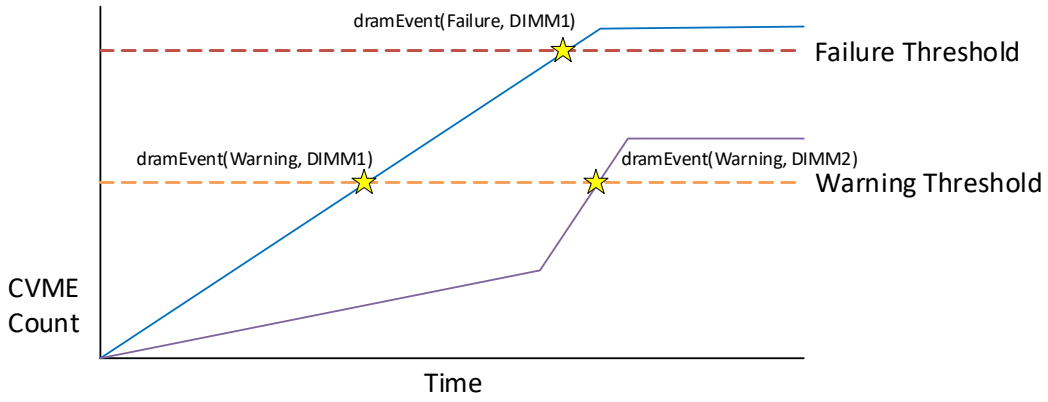


Figure 2 - Example of per Memory Media FRU (DIMM) Granularity

Corrected Volatile Memory (CVME) Error Threshold Configuration Flags – The single bit error and multi-bit error masks determines whether those error types detected and corrected by the CXL memory controller will be tracked for thresholding purposes. In some implementations, only tracking and thresholding corrected multi-bit errors may be interesting or useful.

The patrol scrub CVME threshold flag enables the CXL memory device to track and threshold CVMEs detected during patrol scrub in separate counters from those detected during host read/write accesses. This could be useful in determining the number of memory locations with errors, as opposed to potentially counting multiple errors at a single location due to repeating memory access patterns. If the patrol scrub CVME threshold flag is set, then all related patrol scrub threshold settings in this feature should be configured. Additionally, event records based on the patrol scrub CVME threshold are generated with the 'Transaction Type' field set to '05h = Media Patrol Scrub'.

If patrol scrub CVME thresholds are not enabled, then the device counts CVMEs detected during patrol scrub and host accesses together. All remaining patrol scrub related fields are ignored. The 'Transaction Type' field in the event records may be set to '05h = Media Patrol Scrub', '01h = Host Read', or '02h = Host Write' – depending on what transaction occurred at the time the threshold was exceeded.

The counter expiration flag enables this advanced CVME thresholding feature to take on 'leaky bucket' like behaviors. After the programmed timer has expired, the CXL memory device will reset all CVME counters. The device will log General/DRAM events for all subsequent threshold excursions. Care should be taken that programmed CVME thresholds and timer values are thoughtfully set. Extremely low values for either may result in the device 'spamming' the event record log.

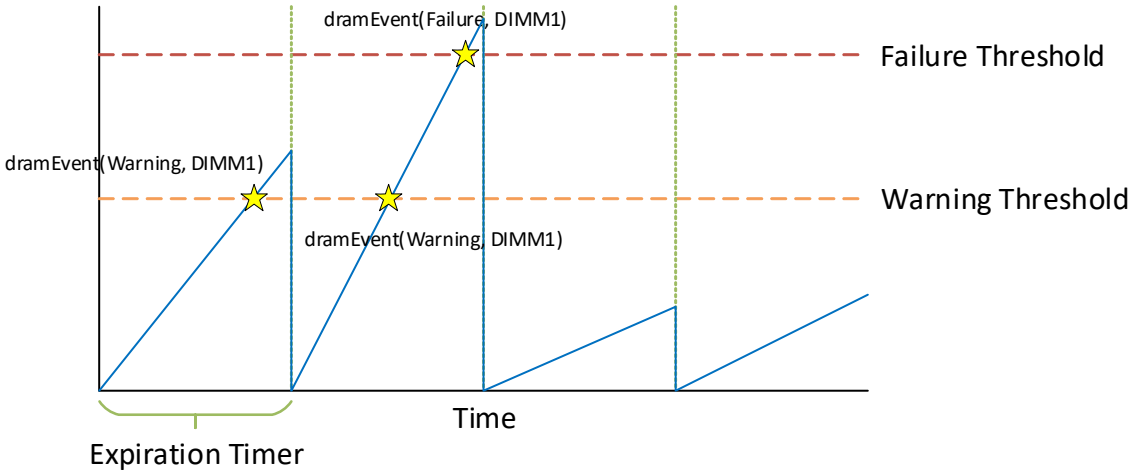


Figure 3 - Example of CVME Counter Expiration (Leaky Bucket)

A related counter expiration reporting flag may also be set, which instructs the CXL memory device to generate an informational General/DRAM event every time the timer has expired. Like previously mentioned, use of this flag should be carefully considered – as this will generate even more events as shown below.

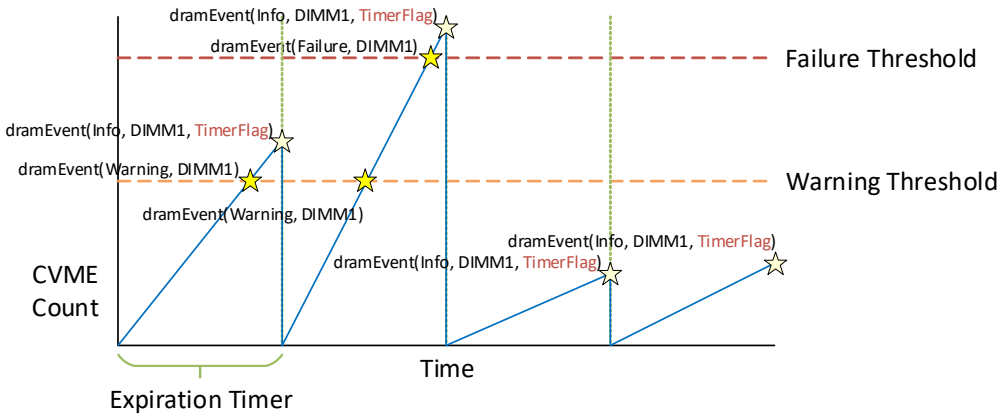


Figure 4- Example of CVME Counter Expiration Reporting

Corrected Volatile Memory Error (CVME) Threshold Event Records Flags – Flags for enabling informational, warning, and failure thresholds allow any combination of these severity thresholds.

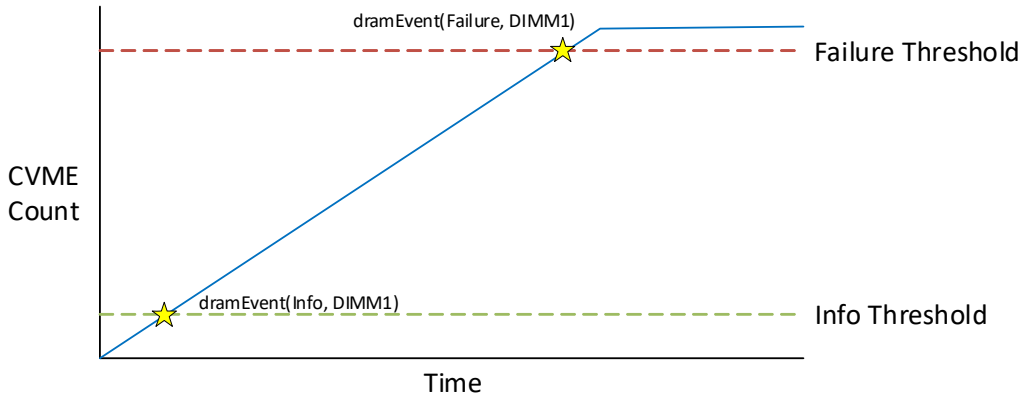


Figure 5 - Example of Info and Failure Thresholds Enabled (No Warning)

Flags for enabling Hardware Replacement Flags configure the CXL memory device to use HW Replacement flags in the event header for warning and failure General/DRAM events generated by this feature.

3.2.2 Changes to General Media & DRAM Event Records for Advanced CVME Thresholds

Several changes have been made to the General Media and DRAM Event Records to accommodate the new Advanced CVME Threshold Feature.

Signaling Advanced CVME Threshold Events

CXL memory devices differentiate the source of CVME threshold event records by use of bit[1] in the 'Advanced Programmable Corrected Memory Error Threshold Event Flag' of General/DRAM event records, as shown below for the DRAM event record:

CVME Thresholding Source	Memory Event Descriptor (Offset 38h, Bit[1])	Advanced Programmable Corrected Memory Error Threshold Event Flag (Offset 7Ah, Bit[1])
Set Alert Config, Internal Device Threshold	Set	Not Set
Advanced CVME Threshold	Set	Set

CVMEs in Multiple Memory Media Components

If the CXL memory device has detected CVMEs in multiple memory media components (e.g., DRAM components), bit [0] of the 'Advanced Programmable Corrected Memory Error Threshold Event Flag' will be set. This can help with determining a need for further error maintenance actions or additional debug – as the General and DRAM event records will only return the location of the last CVME at the time of the threshold being exceeded.

CVME Count at Event

Threshold events records are typically only created when the count of CVMEs has exceeded the programmed threshold. For example, a warning threshold of 100 means that an event is generated at the 100th CVME. However, with the introduction of the expiration timer reporting function in the Advanced CVME Threshold Feature, information threshold event records can be generated at the

moment of timer expiration. It may be useful to know what the actual CVME count was at the time of expiration. Therefore, this field was added to support this level of debug.

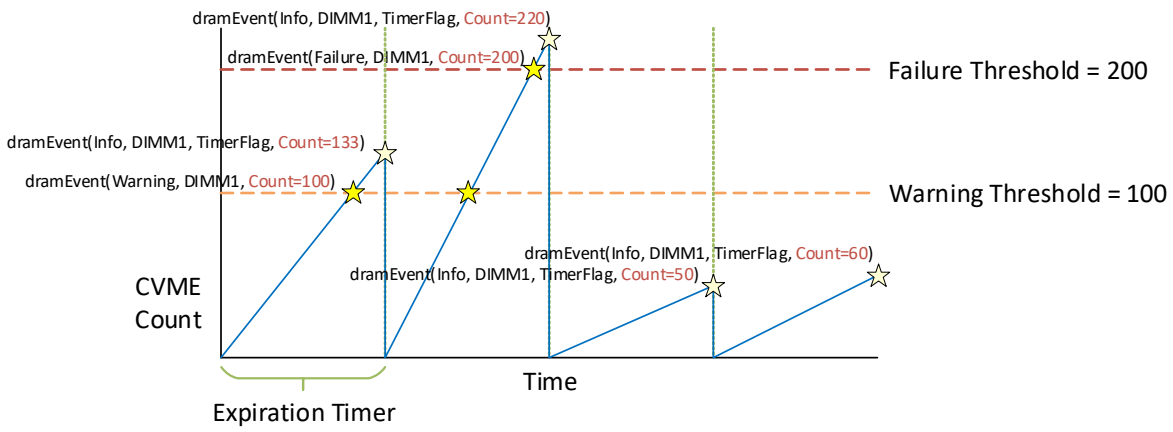


Figure 6 - Example of CVME Counter Expiration Reporting with CVME Count

3.3 PPR and Memory Enhanced Sparing Operations

The first feature that could use a maintenance operation was PPR (Post Package Repair). This feature was introduced in CXL 3.0. It is meant for CXL devices to perform repair operations on their DRAM components. There are two versions of this feature: Hard PPR (hPPR) that is a permanent row repair and Soft PPR (sPPR) that is a temporary row repair.

Later a more generic feature definition was implemented and the DDR enhanced memory sparing was added in CXL 3.1. This enhanced sparing capability was created to encompass the different scopes of sparing available for DDR memory devices.

In this section we will describe the use of these features, as well as compare their inputs and outputs to clarify when each should be used.

The sparing features have a very similar input/output strategy to that first introduced for PPR. The input for the sparing features consists of all the components that conform the address to the spare memory region. In PPR the addresses are provided through DPA (Device Physical Address); while in the enhanced memory sparing, the address follows the DDR Event Record format. The enhanced sparing chooses this format based on the assumption that any sparing decision should come from a predictive failure analysis system that is fed from those error records. Both systems also accept the nibble mask to isolate a specific device for sparing. So, in every scope level (for example row sparing) the sparing functionality might support the sparing of a single device within the scope and therefore the nibble mask will help identify the device. Additionally, both systems have a flag to determine if an actual sparing is being requested or just a query for resources.

One difference between PPR and enhanced memory sparing is the many scopes that are possible in enhanced sparing: cacheline, row, bank, and rank, while the traditional PPR is only limited to row sparing. It is important to note that PPR is the most common and widely deployed sparing mechanism for DDR today. Another difference is that PPR separates the hard (permanent) from the soft (temporary) repairs

by means of a separate subclass, while the enhanced memory sparing uses a flag to hint the device on the preference for the sparing.

A further difference is in the way that the feature handles the restriction flag. The PPR feature has one bit to show that the device remains responsive through the sparing and another bit to highlight if it preserves the data after the sparing. Since the only sparing use case found today use those two flags in a single way; that was condensed into a single flag in the enhanced memory sparing; providing a single bit to highlight if the device will preserve the data and be responsiveness to CXL.mem traffic or if otherwise will not be responsive and loose the data after the sparing. Also as mentioned before instead of creating more subclasses the soft/hard options were added as other restriction flags that can be queried.

Finally, both sparing mechanisms use the new memory sparing event record to report on the success or failure of the sparing request. This new event record format was added in CXL 3.1 and provides the basic information that allows the host or management agent to determine if the sparing was successful or not. It is also used when querying for sparing resources; when that flag is set the sparing does not happen, but the device returns the feasibility of it happening at that address. The number of resources left in the device for sparing at a similar address are expressed as a 2-bit flag “Spare Resource Available”.

3.4 DDR5 Error Check Scrub Control

The CXL 3.1 Specification introduces the control feature of Error Check Scrub (ECS), defined in JEDEC DDR5 DRAM Specification (JESD79-5). Using Error Correcting Code (ECC) on-die, ECS allows the DRAM to internally read, correct single-bit errors, and write back corrected data bits to the DRAM array while providing making error counts available. ECS control allows the configuration of ECS parameters. The feature attributes are a pass through to the JEDEC defined ECS configuration parameters. The readable and writable attributes are defined on a per FRU basis so there is a set of parameters for each FRU. The following definition uses the JEDEC ECS definition of the parameters and the CXL definition to explain how to manage each attribute:

- DDR5 ECS log capabilities: log entry type of how the log is reported, either per DRAM or per memory media FRU. This type is uniformly defined across all memory media FRUs within the CXL device.
- ECS specific parameters per memory media FRU:
    - ECS threshold count per Gigabit of memory cells: 256 (default as defined in JEDEC DDR5 specification), 1024 or 4096.
    - Codeword/row count mode: ECS can either count rows with errors, or codewords with errors.
    - ECS reset counter: when this configuration bit is set to 0, ECS counter runs actively, allowing normal ECS operation. When this configuration bit is set to 1, ECS counter and all other ECS related information such as DRAM row address with maximum number of errors, the maximum number of errors for that row, and the error count are reset to default values. This means that the ECS counter is automatically clear to 0 by the CXL device after the completion of the reset.

It should be noted that JEDEC DDR5 specification also defines ECS mode (manual or automatic), but for CXL 3.1 specification the CXL device is responsible to handle the ECS mode accordingly, depending on the

DDR5 DRAM status and commands that need to be issued by the CXL memory controller within the CXL device.

DDR5 ECS log consists of the following information for the host to observe the results of the ECS operation:

- Common header: this header includes the total number of log entries, the format of Component ID (vendor specific or PLDM-based component ID format defined in CXL 3.1 specification), and the log entry type (per DRAM or per memory media FRU). The total number of entries depends on the number of memory media FRUs, if the log entry type is per memory media FRUs. If the log entry type is per DRAM, the total number of entries also depend on the number of monolithic DRAM dies for each memory media FRU. The log entry type is defined commonly for all memory media FRUs within the CXL device.
- Log fields per entry:
    - Component ID: used to identify a field replaceable sub-component associated with the entry.
    - DDR5 ECS specific parameters such as ECS threshold counter per Gigabit of memory cells, and codeword/row count mode.
    - Error count and address information: indicates whether
        1) Error exists in the DRAM die and a valid log that is required to be read by the host exists.
        2) Error count exceeding the ECS threshold count where the count may differ based on codeword or row count mode.
        3) The number of errors within the row with the most errors.
        4) The address with the highest error count.

Figure 7 shows a flow chart that describes how the host can discover the DDR5 ECS capabilities from the CXL device, how the host can configure ECS capabilities of the CXL device, how the DDR5 ECS operation flow is executed, and what information are reported to the host via ECS log. The host can retrieve whether the ECS is supported by the CXL device, and then retrieve the ECS configuration data that describes the ECS log capabilities, ECS reporting capabilities, and ECS configurations. After the retrieval of ECS configuration data, the host can modify the configuration as needed, or retrieve the ECS log upon its polling or interrupt cycle, where how the ECS log is accessed by the host is outside the scope of the CXL specification.

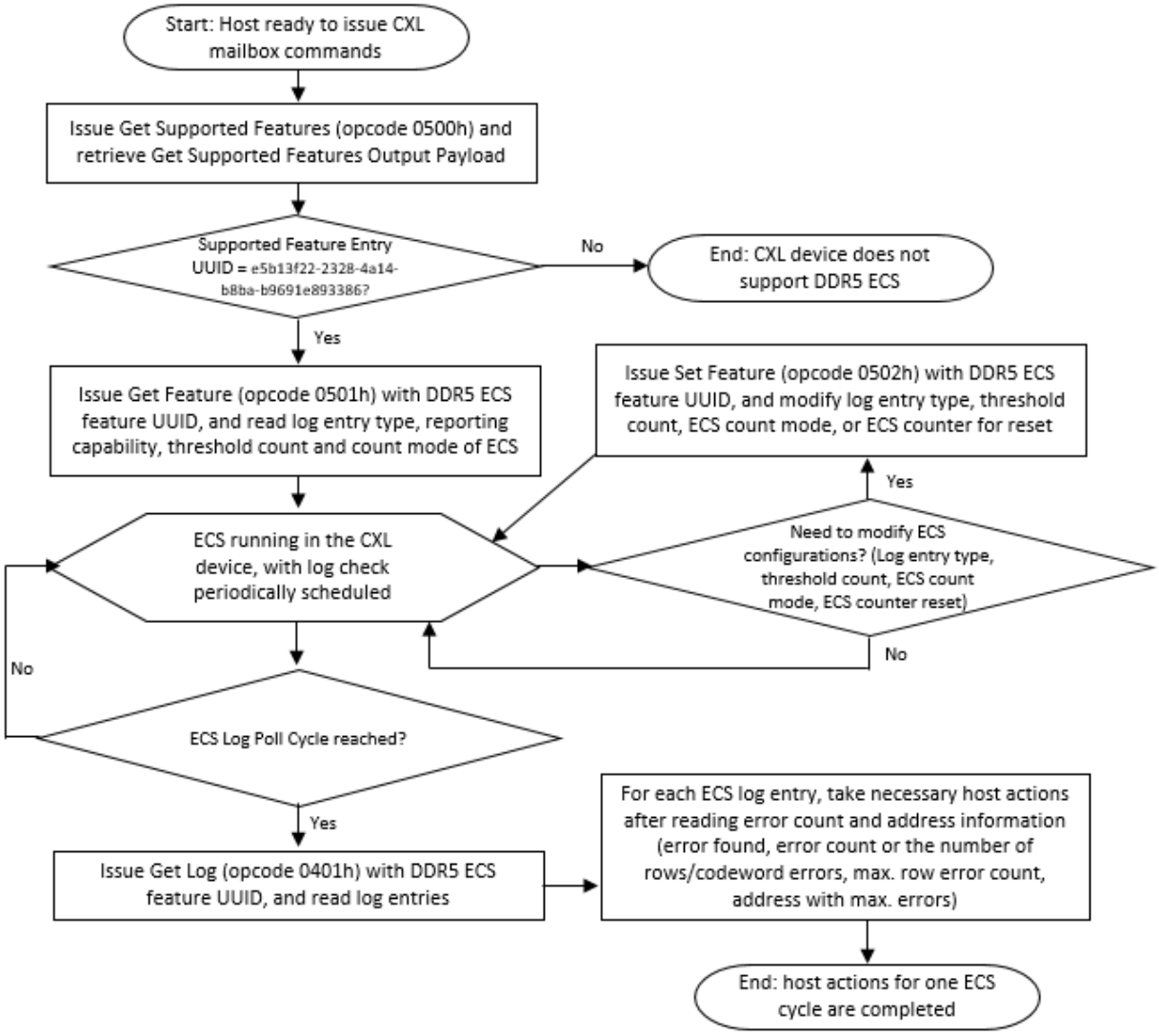


Figure 7 - Flow Chart of the Host Execution of DDR5 ECS Operation of the CXL Memory Device

Although there are multiple ways to implement the host engine dealing with DDR5 ECS, when the DDR5 ECS control feature in CXL devices is used the host is highly recommended to utilize the similar system software infrastructures that were used when accessing DDR5 MR (Mode Register) data, meaning that how the CXL mailbox is accessed should be consistent with how the system software accesses DDR5 MR.

In addition to configuration parameters and log, the ECS control feature allows the host to discover ECS capability of real-time reporting. If the real-time reporting capability is used, the scrub events are generated as the CXL device finds errors in real-time. If the real-time reporting is not supported or enabled, then scrub events are generated only at the end of the scrub cycle, which is 24 hours by default in DDR5. Per definition in JEDEC DDR5 specification, real-time reporting is not supported by DDR5 ECS, but ECS with other DRAM technologies may have such capabilities in the future.

For more information on ECS specific parameters, please refer to mode register and feature descriptions in JEDEC DDR5 specification (JESD79-5).

3.5 Device Patrol Scrub Control

The device patrol scrub control allows the host to configure patrol scrub configurations during system boot or at runtime, proactively locating and making corrections to correctable errors during normal device operation. Patrol scrub control feature introduces the capability for the host to discover:

- whether patrol scrub feature is enabled: this allows the use case when the host wants to verify that it stops the patrol scrub for any reason.
- whether the patrol scrub cycle can be changed upon the host request: Recognizing that changing the patrol scrub cycle time in run time is desirable, especially in debug scenarios; the knob to show if that is supported by the hardware vendor was added.
- whether the patrol scrub event record generation and reporting are supported at real-time: it is highly desirable for hardware vendors to provide a report for each error that is found (corrected or not), but it is not mandatory; and as such these knobs provides the place to highlight it.
- the number of hours for the configured patrol scrub cycle: this is the setting that is currently being used by the patrol scrub to check the memory.
- the smallest number of hours for the patrol scrub cycle supported by the device: this information provides the lower end limit of the time setting for the patrol cycle.

In addition, the patrol scrub control feature allows the host configuration of the scrub parameters as follows:

- Device patrol scrub cycle change: a nonzero value within the range of numbers greater than or equal to the smallest number of hours for the patrol scrub cycle supported by the device.
- Device patrol scrub enable: this bit can be set to 0 to disable the patrol scrub or set to 1 to enable the patrol scrub.

Compared to the DDR5 ECS control feature where a separate log is managed, the patrol scrub control feature uses either the General Media Event Record or the DRAM Event Record when the number of memory errors detected exceeds the configured threshold. Furthermore, the patrol scrub has a larger error handling coverage compared to the DDR5 ECS, where the patrol scrub is expected to handle both uncorrectable errors and corrected errors and the DDR5 ECS handles only corrected single-bit errors.

3.6 Device Built-In Test and Abort Command

Device Built-in Tests are launched issuing Perform Maintenance command. CXL 3.1 defines tests related to media, and other types of tests will be added in future versions of the standard. The media can be tested using several algorithms: Write-Read-Compare, Checkerboard, MARCH, MATS (Modified Algorithmic Test Sequence), etc. Device built-in test may be initiated only if there is no CXL.mem traffic and no CXL.mem requests shall be sent during the test execution. One or more tests can be launched issuing a single Perform Maintenance command. Tests are executed sequentially.

- Device media testing capabilities can be discovered by retrieving Media Test Capability Log, which indicates the number of supported tests, if metadata bits are included in the test, and if

testing of metadata bits and ECC bits can be enabled or disabled. For each test, the log also indicates the maximum test execution time per GB and several other capabilities: if the address range is configurable, if the test can be repeated complementing the pattern, if the test can be stopped on the first uncorrectable error, if an error count threshold is supported, if the Poison list is updated upon uncorrectable errors, the addressing mode (ascending, descending, etc.), the pattern (user provided, PRBS (Pseudorandom Binary Sequence) , 55h, AAh, etc.), and the PRBS length.

The following testing algorithms are defined:

- Write, Read, and Compare
- Checkerboard
- MARCH
- MATS
- MATS+
- Walking 1s
- Walking 0s

Media tests are initiated by issuing Perform Maintenance Command with

- Maintenance Operation Class = 03h (Device Built-in Test)
- Maintenance Operation Subclass = 00h (Media Test)
- Test parameters = Configuration parameters for the tests to be executed.

The Test Parameters may be fully transferred in a single command or transferred in multiple chunks by issuing multiple Perform Maintenance commands with the proper setting of the Action field.

There should be no CXL.mem traffic during test execution.

Media tests may be performed in background and the progress of the operation can be checked reading the Background Command Status Register. When tests are completed, Media Test Results Short Log and Media Test Results Long Log provide information related to the tests that were executed.

For each test, information returned by Media Test Results Short Log includes:

- Test ID
- Start time, Stop time.
- Result (Completed with success, completed with failure, Aborted)
- Uncorrectable Error Count, Correctable Error Count
- etc.

In addition to what returned by the Media Test Results Short Log, the Media Test Results Long Log indicates:

- Capacity that has been tested
- For each DPA in error, an Error Signature with information about the error: DPA, Channel, Sub-channel, Rank, Row, Column, Nibble Mask, Correction Mask, etc.

If errors are detected during media test, a device may be repaired performing PPR maintenance operations or memory sparing maintenance operations. If a maintenance operation Feature has Device Initiated bit set, the device may perform a repair operation automatically.

A component may implement a Mailbox Register CCI and a MCTP-based CCI. Each CCI can support the execution of only one background command at a time. If a component supports Perform Maintenance command on both CCI, and a media test is ongoing in one CCI, an attempt to initiate a new media test on the other CCI will be terminated with Busy return code.

As previously described, Media tests could be executed in background. If a high priority access or an operation need to be performed on the device, the media test may be aborted issuing a Request Abort Background Operation command. The command must be issued on the same CCI as it was invoked. This can be particularly relevant for media tests that can have a long execution time.

4 RAS Enhancements from CXL 2.0

After the release of CXL 2.0 several section pertaining to RAS have been updated and enhance in order to clarify or add RAS functionalities to the CXL devices. In this section the rational for the changes and its intended use are described.

4.1 Changes to Event Records

Event records added some functionality as a support for the other features and RAS functionalities that were added in CXL 3.1.

In the common event record the maintenance operation subclass was added in order to help distinguish if the event record comes from a maintenance operation which subclass of maintenance it belongs to.

One of the most important additions to the event records is the inclusion of the component identifier. The component identifier is explained in the previous section of this whitepaper.

As part of the new features to better manage correctable errors two new fields were added to the general event record: “Advanced Programmable Corrected Memory Error Threshold Event Flags,” “Corrected Memory Error Count at Event.” These fields are discussed in the chapter about “Advanced CVME threshold.”

Finally, more granularity and options have been added to the “Memory Event Type” and the “Memory event subtype” which are aimed to extend possible sources of errors and specify some debug information on where the error happen on the device.

4.2 Memory Capacity Degradation Flows

This feature solves what must happen in a system when a portion of the CXL memory hits an unrecoverable failure. Prior to CXL 3.1 a failure for this case would result in no MEM_ACTIVE assertion and the CXL.mem card would not enumerate as memory. For this case, the device may map out the DIMM, memory channel or a rank. This map-out will cause a memory capacity degradation, performance degradation, or both. A memory degradation is defined as when Media has lower capacity or lower performance than the manufacturer has originally stated. Two new HW register bits were added for this feature.

1) Update **Get Health Info Output Payload** Health Status.
    - Memory Capacity Degraded Bit[3] added to communicate to host that a device in system is operating with a lower memory capacity than manufacturer specified. Health Status changes should propagate an event to the Warning Event log.

Update to Table 8-100 Get Health Info Output Payload as follows

Byte Offset	Length in Bytes	Description
00h	1	**Health Status:** Overall device health summary. Normal health status is all bits cleared. • Bit[0]: Maintenance Needed - The device requires maintenance. When transitioning from 0 to 1, the device shall add an event to the Warning or Failure Event Log, update the Event Status register, and if configured, interrupt the host. • Bit[1]: Performance Degraded - The device is no longer operating at optimal performance. When transitioning from 0 to 1, the device shall add an event to the Warning Event Log, update the Event Status register, and if configured, interrupt the host. • Bit[2]: Hardware Replacement Needed - The device should immediately be replaced. When transitioning from 0 to 1, the device shall add an event to the Failure or Fatal Event Log, update the Event Status register, and if configured, interrupt the host. • Bit[3]: Memory Capacity Degraded - The device is operating at a lower memory capacity than the manufacturer specified capacity. When transitioning from 0 to 1, the device shall add an event to the Warning Event Log, update the Event Status register, and if configured, interrupt the host. • Bits[7:4]: Reserved

Figure 8 - Newly updated bits in Health Info Output Payload

2) Updated Tables 8.1.3.8.2 **DVSEC CXL Range 1 Size Low** and 8.1.3.8.6 **DVSEC CXL Range 2 Size Low**.
   - Memory_Active_Degraded Bit[16] added to both registers indicating when the memory is active but operating in degraded mode.
   - Bits to be set when Media is ready for use when a degradation happens after Memory_Info_Valid=1.

Bit	Attributes	Description
..	..	..
16	RW	Memory_Active_Degraded: When set, indicates that the CXL Range 2 memory is initialized and available for software use regardless of whether the device implements CXL HDM Decoder Capability registers. However, the memory is operating in a degraded mode. If this bit is 1, Memory_Active in this register shall be 0. If Memory_Active in this register is 1, this bit shall be 0. Either Memory_Active or Memory_Active_Degraded must be set within Range_2 Memory_Active_Timeout of reset deassertion to the CXL device when Mem_HwInit_Mode=1 in the DVSEC CXL Capability register.
..	..	..

Figure 9 - Newly updated bits in DVSEC CXL Range Size Low

System Firmware/OS is required to recognize updated Health Messages based on Event logging or to perform a re-read of updated CDAT device capabilities (i.e., new memory map creation).

4.2.1 Capacity Reduction Scenarios and their Handling

CXL devices may detect unrecoverable errors during initialization/test causing it to operate in a degraded mode.

4.2.1.1 Baseline: No Degradation, Media Ready

No memory capacity degradation has occurred, the system uses CXL devices with expected memory size. Device reports correct memory size from Memory_Size_* valid after Memory_Info_Valid=1. If Mem_HwInit_Mode=1, Memory_Active is set after initialization indicating that Media is ready for use.

4.2.1.2 Case 1: Degradation before Memory_Info_Valid==1

Memory_Info_Valid=1 indicates to the system that the Memory_Size_* is ready for memory map creation. Memory Degradation before Memory_Info_Valid implies device able to modify the memory size prior to it being consumed by the system software.

If a failure is detected *Prior to* Memory_Info_Valid=1 causing the capacity to be changed, the original Memory_Size_* is invalid. If errors are detected at this stage, the device shall update the corresponding Memory_Size_* fields for each supported HDM range, DVSEC CXL Ranges, CDAT DSMAS structure, response to Identify Memory Device command and response to Get Partition Info command, if necessary, to report the reduced size.

The device also must set the Memory Capacity Degraded flag in the Health Status Bit[3]. If the failure results in performance degradation, update the CDAT DSLBIS structure, and Performance Degraded Flag in the Health status Bit[1] field should be set informing the management software that the device is operating in degraded mode. The event record should be updated with the DeviceID of what was mapped out for the host to understand.

If Mem_HwInit_Mode=1, Memory_Active shall be set, indicating that the usable Media is available for use. Since the memory failure was detected prior to Memory_Info_Valid being set, the Memory_Active bit is the indication for Media ready.

This flow provides the updated memory size for map creation, setting of the Health Status bits, and proceeds with Media Ready after the Memory_Active bit is set.

4.2.1.3 Case 2: Degradation after Memory_Info_Valid=1 and Prior to Memory_Active=1

This case covers the Media not fully completing initialization or Memory Test fails resulting in capacity degradation prior to Memory_Active=1. When memory degradations happen after Memory_Info_Valid=1, there is a chance that the Memory_Size_* fields have been consumed by BIOS/SW with address map creation in progress.

If capacity has been lost, Device Scoped EFI Memory Type Structure (DSEMTS) entries in CDAT shall mark bad memory as “EFIUnusableMemory” (refer to Figure 11) indicating to the SW that it must not use this memory. Refer to the latest CDAT Specification on the UEFI.org main site.

CDAT table needs to be modified to store the usable Memory capacity. To indicate that CDAT structure has changed, device shall increment the CDAT sequence number (refer to Figure 10).

Field	Byte Length	Byte Offset	Description
/*Sequence*	4	12	The contents of CDAT returned by a component may change during runtime. A component shall reset the sequence number to 0 upon reset. Sequence number field shall be incremented by 1 if the content of CDAT being returned is different from the content that was returned last. The sequence field shall roll over after reaching its maximum value.

Figure 10 - CDAT Table Describing Sequence Change

Field	Byte Length	Byte Offset	Description
EFI Memory Type and Attribute	1	5	0 – EfiConventionalMemory 1 - EfiConventionalMemory Type with EFI_MEMORY_SP Attribute 2 – EfiReservedMemoryType

Figure 11 - Description of EFI Memory Type and Attribute Table

Indicating to Host that a degraded issue occurred, Memory Capacity Degraded flag in Health Status must be set. If failure results in performance degradation, CDAT DSLBIS structure must be updated (Latency/BW info), and Performance Degraded Flag in the Health status shall be set. If needed, an interrupt could be generated to the Host indicating that a degradation occurred.

Get Partition Info shall be updated reflecting the correct Media capacity.

If Mem_HwInit_Mode=1, device shall set Memory_Active_Degraded bit when Media is ready. The Memory_Active and Memory_Active_Degraded bits cannot both be set at the same time.

This flow provides a re-write of CDAT table with revision sequencing, indicating to Host that a CDAT change occurred. Previously allocated memory is marked as “EFIUnusableMemory” with Health Status written and GetPartitionInfo updated reflecting Media capacity. Memory_Active_Degraded set indicates to the system that memory is ready, albeit in a capacity degraded state.

Host SW will re-read the CDAT making the necessary SW/OS adjustments. Due to backwards compatibility, if memory fails at this stage on earlier CXL revisions, legacy SW will timeout – mapping out the entire device and never setting Memory_Active.

4.2.1.4 Case 3: Degradation after Memory_Active=1 or Memory_Active_Degradation=1

For cases where memory fails/capacity degradation is needed after Memory_Active* is set, systems should utilize existing proactive mechanisms such as sparing, OS page off lining, etc. to keep them functional. Runtime managed RAS processes must be followed to handle these errors gracefully.

5 RAS Flows Clarifications with Recipes and Examples

This section presents specific use cases that have been questioned or debated in the CXL consortium working groups. The use cases here are characterized for showing the less obvious uses of RAS features that and it strives to provide some clarity as to how are they meant to be implemented.

5.1 Discussion on Event Record Severity Error Reporting

The CXL 3.1 specification has little description on the criteria to determine in which event log to place a particular event. The guidance provided in the specification is embedded in places like the in the Get Health Info output payload (Table 8-133) where the following conditions are highlighted to create an event in a particular event log:

- Maintenance needed: Creates an event in Warning or Failure logs.
- Performance degraded: Creates an event in Warning log.
- Hardware replacement needed: Creates an event in Failure or Fatal logs.
- Memory Capacity degraded: Creates an event in Warning log.
- Media status (defined Get Health Info values from Section 8.2.9.9): when not normal; it must create an event in the Failure or fatal logs.

The decision of which event log to use for any error lies with the hardware vendor; but here we provide an optional guidance that might help hardware vendors and software developers align.

Informational Event Log: This event log is the one most likely to be polled rather than having an interrupt assigned to it. We recommend adding only events that provide information to the platform; but that are not expected to trigger any action. For example, the specification calls out for temperature events when it is transitioning into Normal from a different state. Another example can be when reporting the outcome of a maintenance command; that event is informational and most probably not a cause for further action. For this type of event, it is expected that the host will only log the event.

Warning Event Log: This event log should contain information about events that might require actions; but are not time critical. An example of this category is correctable errors; they are used by the host to feed into Predictive Failure Analysis algorithms that will eventually trigger actions; but they are not directly actionable. The host is expected to log and distribute these errors to the corresponding agent for appropriate handling.

Failure Event Log: This event log should include any event that requires immediate attention from the host. Events that are part of recovery flows should be placed here. It is also expected that this queue is always set to use interrupts.

Fatal Event Log: This event log should contain events that are mostly use for root cause of fatal issues. It is expected that events in this log will be followed by a reset due to the severity or recoverability of the event.

5.2 CXL Event Record Encapsulation in CPER Format

The UEFI specification 2.9 introduces a novel method for encapsulating CXL component events. This encapsulation is achieved through a new section type titled "CXL component Events," which is detailed in Appendix N of the UEFI specification.

The section type GUID in the CPER record's section descriptor references the first field from the CXL event's common event record format. It is important to note that the CXL UUID is transformed to a CPER GUID, where a GUID byte swaps the first double-word (32-bit), the second word (16-bit) and third word compared to a UUID. The format of the CPER record mirrors the format of CXL event record after offset 0x10, i.e. the UUID that was translated into the CPER section type GUID is truncated leaving the remainder of the CXL record format as the CPER record format.

5.3 Example Advanced CVME Threshold Implementations

The following example implementation sets up the following:

- Tracks CVMEs at a per Memory Media FRU (DIMM) granularity
- Masks single bit errors
- Leaks / resets the CVME counters every 10 mins.
- Sets the warning threshold = 128 CVMEs
- Sets the failure threshold = 1024 CVMEs with HW replacement flag
- Disables all other features (expiration reporting, patrol scrub threshold, info threshold, etc.)

Byte Offset	Setting
00h	**Configured Corrected Volatile Memory Error Threshold Granularity** 01h = Per Memory Media FRU Granularity
01h	**Configured Corrected Volatile Memory (CVME) Error Threshold Configuration** Bit[0] = 1 : Enable Single Bit Error Mask Bit[1] = 0 : Disable Corrected Multi Bit Error Mask Bit[2] = 0 : Disable Patrol Scrub CVME Threshold Bit[3] = 1 : Enable Counter Expiration Bit[4] = 1 : Enable Counter Expiration Reporting
02h	**Configured Corrected Volatile Memory Error Counter Expiration Timer** 000258h = 600 seconds (10 minutes)
05h	**Configured Corrected Volatile Memory Error Threshold Event Records Flags:** Bit[0] = 0 : Disable Programmable Informational Threshold Bit[1] = 1 : Enable Programmable Warning Threshold Bit[2] = 1 : Enable Programmable Failure Threshold Bit[3] = 0 : Enable Programmable HW Hardware Replacement Flags for Warning Events Bit[4] = 1 : Enable Programmable HW Hardware Replacement Flags for Failure Events
06h	**Programmable Corrected Volatile Memory Error Informational Event Threshold** 000000h = Disabled
09h	**Programmable Corrected Volatile Memory Error Warning Event Threshold** 000080h = 128 CVMEs
0Ch	**Programmable Corrected Volatile Memory Error Failure Event Threshold** 000400h = 1024 CVMEs
0Fh	**Configured Patrol Scrub CVME Threshold Event Records Flags** Bit[0] = 0 : Disable Programmable Informational Patrol Scrub Threshold

	Bit[1] = 0 : Disable Programmable Warning Patrol Scrub Threshold Bit[2] = 0 : Disable Programmable Failure Patrol Scrub Threshold Bit[3] = 0 : Disable Programmable HW Hardware Replacement Flags for Warning Patrol Scrub Events Bit[4] = 0 : Disable Programmable HW Hardware Replacement Flags for Failure Patrol Scrub Events
10h	**Programmable Patrol Scrub CVME Informational Event Threshold** 000000h = Disabled
13h	**Programmable Patrol Scrub CVME Warning Event Threshold** 000000h = Disabled
16h	**Programmable Patrol Scrub CVME Failure Event Threshold** 000000h = Disabled

With the above example implementation, consider the following example CVME situation:

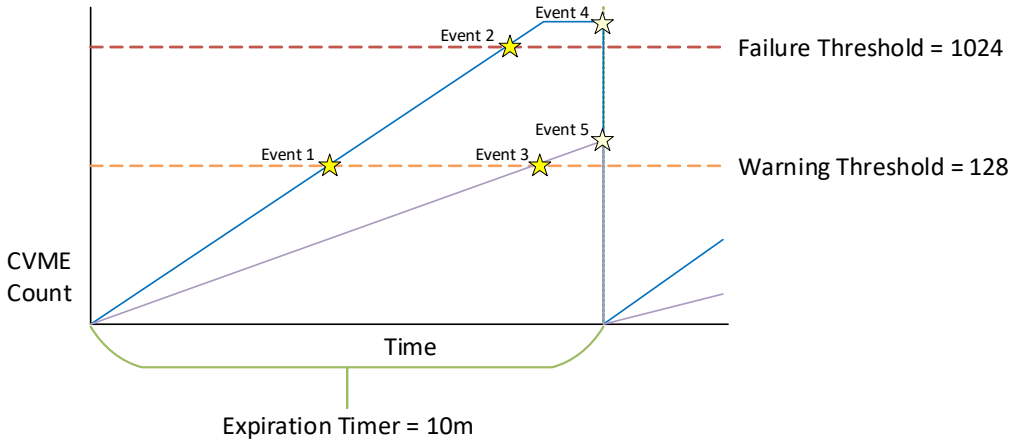


Figure 12 - Example CVME Situation

Five DRAM (or General Media) event records would be generated from this situation:

- Event 1
  - Event Record Severity = Warning
  - HW Replacement Needed = Not Set
  - Memory Event Descriptor = CVME Threshold Event
  - Component ID = DRAM on DIMM 1
  - Advanced Programmable CVME Flags = Threshold exceeded bit[1] set
  - CVME Count = 128
- Event 2
  - Event Record Severity = Failure
  - HW Replacement Needed = Set
  - Memory Event Descriptor = CVME Threshold Event
  - Component ID = DRAM on DIMM 1
  - Advanced Programmable CVME Flags = Threshold exceeded bit[1] set
  - CVME Count = 1024

- Event 3
    - Event Record Severity = Warning
    - HW Replacement Needed = Not Set
    - Memory Event Descriptor = CVME Threshold Event
    - Component ID = DRAM on DIMM 2
    - Advanced Programmable CVME Flags = Threshold exceeded bit[1] set
    - CVME Count = 128
- Event 4
    - Event Record Severity = Informational
    - HW Replacement Needed = Not Set
    - Memory Event Descriptor = CVME Threshold Event
    - Component ID = DRAM on DIMM 1
    - Advanced Programmable CVME Flags = Threshold exceeded bit[1] not set
    - CVME Count = Some number higher than 1024 (e.g., 1088)
- Event 5
    - Event Record Severity = Informational
    - HW Replacement Needed = Not Set
    - Memory Event Descriptor = CVME Threshold Event
    - Component ID = DRAM on DIMM 2
    - Advanced Programmable CVME Flags = Threshold exceeded bit[1] not set
    - CVME Count = Some number higher than 128 (e.g., 134)

5.4 Triggering PPR Actions based on Advance Thresholds.

A very common scenario for memory sparing is the triggering of row sparing using correctable error thresholds. In this scenario we do not need to assume form where the controlling host is issuing the commands; all we need is to have the host or fabric manager control the respective mailbox and have interrupts enabled for the Failure Event queue. A common scenario would be a host controlling a CXL device from the OS and using the primary mailbox to issue these commands; but another scenario is a fabric manager using the CCI interface to control the CXL device from out of band. These second scenario might be more common when the CXL device is not only dedicated to a host and therefore the RAS management is best exercised from the fabric manager that always has visibility of the device. During this example we will use the word host to refer to the controlling agent and interrupt to refer to the interrupt mechanism for the event queues; but they can be the fabric manager and the MCTP interrupt message as defined in the CCI.

The use of the Advance Programmable Corrected Volatile Memory Error Threshold feature will be demonstrated throughout this example. Once the triggering of the sparing happens; two options will be highlighted: using the PPR maintenance feature (soft or hard will be indicated in the example); and the Enhance memory sparing using the row sparing options.

The general flow assumes that the following features are available in the target device as per Get Supported Features command:

Advanced Programmable Corrected Volatile Memory Error Threshold	1478ad9d-ce00-4733-9db8-f392a4c2d0cc

And either these:

Soft PPR	892ba475-fad8-474e-9d3e- 692c917568bb
Hard PPR	80ea4521-786f-4127-afb1- ec7459fb0e24

Or this:

Row - MemorySparing	450ebf67-b135-4f97-a498- c2d57f279bed

The general flow for managing this event is split into two parts: the setup for the thresholds and the handling of the event once it gets triggered.

5.4.1 Setting up CE Threshold

Sometime during initialization, the host must configure the device to create the interrupt when the threshold of CE exceeds the desired limit. The first step to this scenario is to determine in the Adv Corrected CE threshold features can trigger this action. For this the first instruction issued should be Get feature.

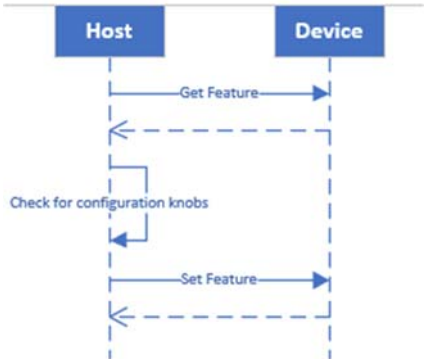


Figure 13 - Setup of Adv Memory Thresholds

The Get Feature command is intended to retrieve the capabilities for the Adv Programmable Corrected Volatile Memory Error Threshold feature. A few capabilities and configuration must be checked to make full use of this feature to its fullest potential. The following are a list of knobs that must be checked first for availability in the device (if the device supports them) and then decide which is the best configuration to use according to the sparing policy for the system.

- **Corrected Volatile Memory Error Thresholds (CVME) Granularity**: This is important to limit the granularity from which the device will be counting errors to signal the event. It could be per Memory FRU granularity (DIMM) if bit [0] is enable; or it can be per Rank granularity if bit [1] is enable. If neither is enabled, then the only option is for the full HDM range.
- **CVME Configuration Flags**: Here we get to know and configure if the device will filter by type of correctable error (single-bit vs multi-bit) or by its source (patrol scrub vs demand scrub). In this example the policy being implemented does not distinguish between different types or source of errors. We can also configure if an event should be issued at the end of a predetermined timer (counter). The counter is interesting since it changes the thresholds from being simple counter

that have no time limit to be reached to a time-based window in which the thresholds must exceed. For the example developed here the timer will be set to 24 hours, so the numbers of errors detected must be greater than the threshold in a single day to trigger the sparing action. It is recommended for this example to set the bit [4] Enable Counter Expiration Reporting to zero so there is no event generated at the end of each counter expiration.

- The **CVME Threshold Event Records**: Here is where the correct severity must be enabled to trigger the event that will cause the sparing. It is not recommended to use the informational severity since it will probably produce many other events and therefore will be hard to filter the right message to trigger the sparing action. In this example we will configure only the Failure event record.
- The **Programmable CVME Event thresholds**: In these set of fields, we must program the actual counter to trigger the threshold event, so we program the Failure event Threshold to the desired policy (one thousand errors for this event).
- The **Configured Patrol CVME thresholds** allows for the same configuration to be done for patrol scrub only counters. In this example the policy being implemented does not distinguish by the source of the error detection, so these fields are not enabled.

Use the set potions as shown in Figure 13 to configure the right parameters for the feature. That will finish the setup and this step of the process.

5.4.2 Handling the Threshold Event

The next step in the process is the handling of the event created by the advance thresholds feature. Let us first get the bigger picture of what needs to happen.

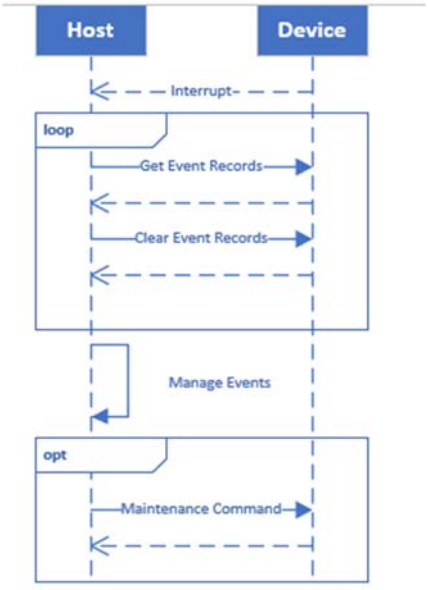


Figure 14 - Handling the Threshold Event

The first loop in the process diagram is depicting the normal use of the mailbox to read the event records from the error queues. This process happens as normal until the specific event that has been configured is found; in the example here is a **memory media event record** of **failure severity**; more specifically one

where the Advanced Programmable Corrected Memory Error Threshold Event Flag (byte offset 52h) has bit [1] set indicating the threshold for this feature has been exceeded. Once this specific event has been found the second part of the action that is the actual command to perform the sparing can be acted.

5.4.2.1 Sparing using PPR Commands

The necessary pieces to issue a PPR command are derived from the error events that got captured. The event generated by the advance thresholds feature, will have the address of the last error that equaled or exceeded the threshold in that granularity. So, for example if the granularity was set to rank level (the lowest granularity that can be advertised) the event represents the 1000th error that happened on that rank. Their needs to be a policy or predictive failure analysis tool that will help determine which row is about to produce an uncorrectable error in that rank and needs to be spared. This analysis is specific to the host controller and can use not only the single failure event record that exceeded the threshold; but the one thousand informational events that were produced before that threshold was exceeded.

Once that row address is identified; the DPA (Device Physical Address) must be obtained; and the desired nibble (if a single DRAM device is suspected faulty within that row). The command that will be used in this example is the sPPR command since this is expected to be a runtime solution; a different flow is needed for hPPR, where the device must perform a reset for the fix to take place. The sPPR command gets issued with the correct address and without the query resources flag (in this example we are not querying for resource availability, but instead issuing the command and verifying the outcome). This is done issuing a **Perform Maintenance** command (Opcode 0600h) with **Maintenance operation class** 01h and **Subclass** 00h to indicate the sPPR action. The result of the RAS action can be immediate; but it is most likely a background operation. After the operation is completed (the host can determine this by checking the registers in the **Mailbox Status Register** or by querying the CCI using the **Background Operation Status (Opcode 0002h)**.

The second alternative we must discuss is the use of the **Enhance Memory Sparing**. In this case the **Perform maintenance** command should use as input the location as expressed by channel, subchannel, rank, bank group, bank and if desired nibble mask. The **class** must be set to 02h and the **subclass** to 01h to specify a enhance memory sparing with the scope of row. The hard sparing flag should be unset (0) as well as the query resource flag. This operation will be a background operation, so the same procedure most be followed to query the outcome of the operation.

After the background operation is finalized; the host must query the information event queue to find the outcome of the sparing. The queue should contain the event in the format of **Memory Sparing Event Record**. Here the host will get confirmation of the address and a result indicating if the sparing was successful. If the sparing is a success (or if the original flow included a query portion) the **Spare Resource Available** field might indicate if there are more resources available at that specific sparing location.

5.5 Viral

This system feature, when enabled, is meant to avoid corrupting contents of permanent media in the virtual hierarchy. It is required to be supported by all CXL devices though it is enabled based on system policy. This feature is applicable only to CXL.cachemem; It is not meant for CXL.io.

As this is a big hammer from system perspective and entire virtual hierarchy will go through reset sequence, it should be used rarely. For most errors, use of Poison should suffice. Using poison for

containment of bad data works in most cases; but for some systems where it cannot be guaranteed that all the devices on the platform can correctly handle poison it is desirable to turn on viral. This is especially important since CXL devices now hold coherent memory and devices that might not be capable of filtering poison can DMA into the memory and spread corrupted data.

To prevent corrupted data from going into storage, CXL devices must self-isolate, which means they must guarantee the containment of corrupted data without any host assistance. The device must be wary that the signal for viral reaches the host before any corrupted data can do it. This is sometimes referred to as the "race condition" where viral signaling must by design beat the corrupted data. On top of containment, the devices should make sure that the host can make forward progress so the cause of viral can be determined and the reset correctly orchestrated. A list of the types of actions the device must perform during viral is clearly laid out in the CXL 3.1 specification chapter 12.4.2.

The viral condition can only be cleared by a conventional reset; CXL resets and FLRs have no effect on the viral condition.

5.5.1 Viral Signaling Flow

This section assumes the viral feature has been enabled at system level. The viral state can be triggered in two ways, the host might encounter viral, or the device might encounter uncorrectable fatal error and trigger viral. If the host gets the viral condition; it will communicate it to all downstream devices before any corrupted data can be committed to permanent storage.

For the second flow, viral feature is triggered when an uncorrected fatal error is detected; at that point, the device enters the viral state.

The Viral bit is sent using RETRY.Ack flit for 64Byte flit mode and using "In-band Error" type Control Message flit for 256-byte flit mode. When Viral is set, along with it, LD-ID vector/field should be set appropriately if fatal error is isolated to a particular LD-ID or a set of LD-IDs. For SLDs or for MLD devices when LD-ID cannot be determined, LD-ID should be set to all ZEROs to indicate that all the LD-IDs are affected.

5.5.2 Viral Control and Status bits are Defined in the PCIe DVSEC for CXL Devices

DVSEC CXL Capability (Offset 0Ah)

Viral_Capable: If set, indicates that the CXL device supports Viral handling. This value must be 1 for all devices.

DVSEC CXL Control (Offset 0Ch)

Viral_Enable: When set, enables Viral handling in the CXL device. Locked by the CONFIG_LOCK bit1.

If 0, the CXL device may ignore the viral that it receives. Default value of this bit is 0.

DVSEC CXL Status (Offset 0Eh)

Viral_Status: When set, indicates that the CXL device has encountered a Viral condition. This bit does not indicate that the device is currently in Viral condition.

5.6 Data Poison Handling

Data poisoning plays a pivotal role in ensuring data integrity and system reliability. By understanding its nuanced operation, system implementers can better mitigate risks associated with uncorrectable memory errors.

Poison is design to contain and manage uncorrectable memory errors, this avoids data corruption. Align to this objective the device shall poison the data message on the CXL interface whenever the data being included is known or suspected to be bad. To keep track of this we normally assume the device sets a poison flag for each cacheline that fails the ECC; but other implementation that change the data contents to a special token that is always assumed bad also work. It is important that when an uncorrectable error is detected; the memory location be mark in some way; to prevent further bit flips to turn the data and ECC into a different valid data. In other words, the ECC algorithms work best with a low number of errors on the same cacheline; if errors are permitted to accumulate indefinitely the ECC algorithm won't be able to distinguish this morphed data from valid data.

It is also important to highlight the basic rules for logging poison. Poison errors should be logged only when they are first discovered in the system or when they are consumed; not when they are transmitted. So, if a cacheline is already marked as containing poison; the device should always transmit the poison flag with the data; but should not log a poison event.

5.6.1 Data Poison Flows

The following are some commonly expected memory poison handling scenarios that must be dealt with by system and OS implementers. Management entities described in this section may be a BMC, system BIOS, system OS, or even a management application.

Poison can originate from either the host or the device itself. Furthermore, for the device poison can be found either during a read operation or a patrol scrub operation. In the following sections the different handling methods are discussed.

5.6.1.1 Poison Write from Host

This flow details the high-level sequence of events when a host writes poisoned data to a CXL memory device. This typically occurs during a cache line eviction, and the data has been previously corrupted during transmission or storage.

1. Host is aware of existing poison and/or detects an uncorrectable error in cache contents. During cache line eviction, the host writes to CXL memory device with poison bit set.
2. The CXL memory controller writes the poisoned data to the appropriate media location with the poison flag.
    a. If poison list is supported, controller adds location to poison list.
3. No memory media or DRAM event records are generated since the poison did not originate at the device.

It is important to note that the same sequence applies to partial writes. If the poison came from the host, the device cannot know which bits are bad. Since there is no way for device to know that the partial write fixed any bad bits it must mark the whole cacheline as poisoned. If the poison was detected previously by the device; there is no way for the device to know that the partial write fixed any cacheline, so that cacheline must remain with the poison flag.

5.6.1.2 Patrol Scrub

This flow details the high-level sequence of events when a CXL memory controller detects an uncorrectable error in the memory media during patrol scrub.

1. CXL memory controller scrubs through memory media and detects an uncorrectable error.
2. CXL memory controller marks the location with poison bit.
    a. If poison list is supported, controller adds location to poison list.
3. CXL memory controller generates media/DRAM event record indicating an uncorrectable error detection during patrol scrub.
    a. Management entity(s) reads event record and generates appropriate log message, may require decoding Component ID to perform sub-FRU isolation (e.g. DIMM-level isolation).
4. **Optional:** To facilitate memory resource off-lining or process recovery/termination, the OS and system's management entity must coordinate to share the Device Physical Address (DPA) of the uncorrectable error event. Then the OS must perform forward address translation from DPA to Host Physical Address (HPA).

There are a couple of ways that CXL controllers can deal with known poison locations when the patrol scrub finds them. One option is not to scrub through them; since poison locations are known to be bad it shouldn't be an option that random errors correct the data. Poison can only be removed if explicitly written to or cleared. In this case the controller won't generate extra events on the same cacheline location after the first discovery.

The second option is to scrub the poison location and create the event record every time the particular cacheline is scrubbed. This can generate multiple events for the same error location and its generally considered error pollution. In this case any filtering of this extra events would be hardware vendor specific.

From a recovery strategy having multiple events generated for a specific failure could represent just noise. If the recovery strategy is to spare the cacheline (for example using PPR); the poison will be removed once the cacheline is cleared and not before; so, reporting on a cacheline that is marked as poisoned doesn't add any value. It is the same with other methods like page offlining; when the OS removes the page; there is no way for the patrol scrub to skip that section, but it shouldn't generate error events after the poison discovery.

5.6.1.3 Poison Consumption

This flow details the high-level sequence of events when a host requests data from the CXL device and the CXL memory controller detects an uncorrectable error while reading from the memory media.

1. Host performs memory read from CXL memory device.
2. CXL memory controller reads from memory media and detects an uncorrectable error.
3. CXL memory controller marks the location with poison bit. (this is called demand scrubbing)
    a. If poison list is supported, controller adds location to poison list.
4. CXL memory controller generates media/DRAM event record indicating an uncorrectable error detection during host read transaction.
    a. Management entity(s) reads event record and generates appropriate log message, may require decoding Component ID to perform sub-FRU isolation (e.g. DIMM-level isolation).
5. CXL memory device returns uncorrected data to host with poison bit.

6. When poisoned data is consumed by Data Cache Unit (DCU) or Instruction Fetch Unit (IFU), a machine check exception is generated.
7. CPU and OS initiates standard process recovery or process termination mechanisms, as used by homogeneous memory (e.g. local DDR).
8. **Optional:** OS and/or other management entity(s) must perform Host Physical Address (HPA) to Device Physical Address (DPA) reverse address translation to correlate the machine check exception to the CXL device and/or its sub-FRUs.

If the host read originates from a CPU pre-fetch, poisoned data may sit in cache for an indeterminant amount of time, thereby stalling step 6 above. In this case the host might take actions based on the event record before the poison is actually consumed (for example page offlining).

It is important to note that unless the device is configured otherwise, the CXL memory controller is expected to generate an event record and appropriate interrupt for every uncorrectable error detected during a host read. If the cacheline was already poisoned; the device will send the poisoned data and the poison flag the same; but will not generate a DRAM event for it.

Also note that the HPA to DPA reverse address translation is CPU supplier specific so please refer to CPU documentation for reference to the flows involved on it.

5.7 Configuring a Device-Initiated RAS Feature

Some maintenance operations have the capability to be device-initiated. The feature system support this in CXL through the Operation Capability and Operation Mode configurations. In this example the Memory sparing maintenance feature will be used to showcase the functionality.

During setup the OS or management entity can query the available features in the device using the Get Feature command. Once it finds the features each feature will be queried to record its configuration and capabilities. In this example the device will have the “Row Memory Sparing” capability available with the option to do device initiated sparing. The readable attributes important for this example are shown in Table 1.

BiteOffset	Length in Bytes	Description
00h	1	**Maximum Maintenance Operation Latency**
01h	2	**Operation Capabilities**: **Bit[0]: Device Initiated Capability: 1** A value of 1 indicates that the device has the capability to initiate a maintenance operation without host involvement. • Bits[15:1]: Reserved
03h	2	**Operation Mode**: **Bit[0]: Device Initiated: 0** A value of 1 indicates that the device may initiate the maintenance operation without host involvement. If cleared to 0, the device shall initiate the maintenance operation only when receiving an explicit maintenance command. If bit[0] of the Operation Capabilities field returns 0, this bit is considered reserved. • Bits[15:1]: Reserved
05h	1	**Maintenance Operation Class**: **02h (Memory Sparing).**
06h	1	**Maintenance Operation Subclass**: **01h (row sparing)**
07h	10	**Reserved**
11h	2	**Restriction Flags**

Table 1 - Readable attributes or Row Sparing with device-initiated capability.

Notice the Device Initiated capability bit that is set to 1 indicating that the device can initiate this maintenance. It is recommended that this feature is not set by default, and that is why the example shows the Operation Mode still in 0 for Device Initiate. It should be the controller entity's prerogative to enable the device to manage the feature autonomously.

As part of the setup negotiation, the controller entity will enable the device initiated in the operation mode. This is done through a "Set Feature" command changing the Operation Mode bit[0] to 1 (Device Initiated). It is important to note that the actual policies used by the device to enact this feature are not dictated by the CXL specification but are left for the hardware vendor to define and parameterize through vendor define methods.

Now that the device is configured it will monitor the conditions for its internal policies to trigger the sparing action. For this example, that uses row sparing; the policy will most likely be related to correctable errors, temperature, or any other leading indicator of a memory failure. This predictive failure engine is part of the internal vendor define policies that the device must have to offer this device-initiated option.

Finally, when the device needs to trigger the feature, it will do so without consulting the host. It will enact the sparing after the trigger condition when the state of the device permits such an action. Even though the device will follow the commands from the host; it is still aware of the current status and must use that to trigger the sparing action at a convenient time. The host is informed of the action through the event records that get generated with the success or failure of the sparing action.

It will be up to the internal policy algorithms that the device will handle any exceptions and make corrections to its predictive algorithms; like for example when a sparing failed repeatedly in a particular row; stop issuing the sparing action and relying on the host to take further action at that location.

5.8 Example of Device Build-In Test with an Abort Command

Media Test Capability Log indicates the characteristics of the media tests supported by the device. Table 2 shows an example for a device that supports three media test algorithms: MARCH, Write-Read-Compare, and Checkboard. The Capability field in the Common Header specifies that metadata bits can be tested, and both ECC and metadata can be disabled during the test.

Byte offset	Length	Description	Value
Common Header			
00h	1	Number of supported tests	3
01h	4	Error Signature List Size	128
05h	1	Media Test Result Long Log version	1
06h	1	Media Test Result Short Log version	1
07h	1	Capabilities	0Fh
08h	8	Reserved	0
Test 1 Media Test Capability Log Entry			
10h	2	Test ID	0001h
12h	1	Algorithm (MARCH) (1)	04h
13h	1	Execution Time (the maximum test execution time per GB is 3 s) (2)	33h
14h	2	Capabilities (3)	009Dh
16h	2	Supported Patterns (4)	0002h
18h	1	PRBS Length	0
19h	7	Reserved	0

Byte offset	Length	Description	Value
Test 2 Media Test Capability Log Entry			
20h	2	Test ID	0002h
22h	1	Algorithm (Write, Read, and Compare pattern) (1)	01h
23h	1	Execution Time (the maximum test execution time per GB is 1.20 s) (2)	C2h
24h	2	Capabilities (3)	017Fh
26h	2	Supported Patterns (4)	
28h	1	PRBS Length	09h
29h	7	Reserved	0
Test 3 Media Test Capability Log Entry			
30h	2	Test ID	0003h
32h	1	Algorithm (Checkboard) (1)	04h
33h	1	Execution Time (the maximum test execution time per GB is 2 s) (2)	23h
34h	2	Capabilities (3)	009Fh
36h	2	Supported Patterns (4)	0002h
38h	1	PRBS Length	0
39h	7	Reserved	0

Table 2 - Media Test Capability Log Output Payload

1. **Algorithm** 00h=Reserved. 01h=Write Read Compare. 02h=Checkerboard. 03h=MARCH. 04h=MATS. 05h=MATS+. 06h=Walking 1s. 07h=Walking 0s.
2. **Execution Time** Bits[3:0] specify the time scale: 1h=10 ms, 2h=100 ms,3h=1 s, 4h=10 s, 5h=100 s, 6h=1000 s. Bits[7:4] specify the maximum operation latency.
3. **Capabilities** Bit[0]: Address Configurable Flag. Bit[1]: Inverse Pattern Support. Bit[2]: Exit on Uncorrectable Error. Bit[3]: Error Count Threshold Programmable.
   Bit[4]: Update Poison List on Uncorrectable Error. Addressing Mode: Bits[5]: Ascending, Bit[6]: Descending, Bit[7]: Algorithm Specific, Bit[8]: Random.
4. **Supported Patterns** Bit[0]: User provided. Bit[1]: Vendor specific. Bit[2]: Generated by a PRBS. Bit[3]: DPA[63:0] value is repeated eight times to cover 64B. Bit[4]: 55h. Bit[5]: AAh.

As indicated by the Capabilities field related to each test, address ranges are configurable, tests can be stopped on the first uncorrectable error, Error Count Thresholds are supported, and Poison List can be updated on uncorrectable error. Address mode can be ascending, descending or random only for Write-Read-Compare test, while it is algorithm specific for the other tests.

In this media test example, two subsequent tests are executed on a 128 GB DPA range starting from the address 0000001000000000h (64 GB). Table 3 shows the Perform Maintenance input payload used to initiate the media testing. The input payload is transferred with a single Perform Maintenance command (Action = Full Transfer).

The Error Signature Configuration bit of Media Test Results Configuration is cleared to 0b, therefore all error signatures related to correctable or uncorrectable error will be included in the Media Test Results Long Log, if any. Configuration Flags are all zero, therefore ECC is enabled for both data and metadata.

The first test is Write-Read-Compare, the entire 128 GB is tested even in case of uncorrectable error, error counter threshold is not used, the addressing mode is ascending, the Poison List is updated on uncorrectable error, and data is generated by a PRBS.

The second test is MARCH, the test is interrupted at the first uncorrectable error, error counter threshold is not used, and the Poison List is updated on uncorrectable error.

Byte offset	Length	Description	Value
00h	1	Maintenance Operation Class (Device Built-In Test)	03h
01h	1	Maintenance Operation Subclass (Media Test)	00h
02h	1	Action (Full Transfer)	00h
03h	4	Offset	0

Byte offset	Length	Description	Value
07h	1	Reserved	0
Common Configuration Parameters			
08h	1	Number of Tests (2)	02h
09h	8	Start Address (64 GB)	0000 0010 0000 0000 h
11h	8	Length (128 GB)	0000 0020 0000 0000 h
19h	1	Media Test Results Configuration [1]	00h
1Ah	1	Configuration Flags [2]	00h
1Bh	Dh	Reserved	0
Parameters for the first test			
28h	2	Test ID (Write, Read, and Compare pattern)	0002h
2Ah	1	Number of iterations	01h
2Bh	2	Flags [3]	0100h
2Dh	2	Pattern Type [4] (PRBS)	0002h
2Fh	1	Pattern Value	0
30h	2	Vendor Specific	0
32h	4	PRBS Seed	0000 0144h
36h	2	Error Count Threshold	0
38h	10h	Reserved	0
Parameters for the second test			
48h	2	Test ID (MARCH)	0003h
4Ah	1	Number of iterations	01h
4Bh	2	Flags [3]	0122h
4Dh	2	Pattern Type [4] (PRBS)	0001h
4Fh	1	Pattern Value	0
50h	2	Vendor Specific	0
52h	4	PRBS Seed	0000 0000 h
56h	2	Error Count Threshold	0
58h	10h	Reserved	0

Table 3 - Example Perform Maintenance Input Payload

1. **Media Test Results Configuration** Bit[0] Error Signature Configuration: 0b = Complete (all the error signatures are reported), 1b = Single error signature. Other bits are reserved.
2. **Configuration Flags** Bits[1:0]: ECC Disablement: 00b = Data ECC is enabled and metadata ECC is enabled, 01b = Data ECC is disabled and metadata ECC is enabled. 10b = Data ECC is enabled and metadata ECC is disabled. 11b = Data ECC is disabled and metadata ECC is disabled. Other bits are reserved.
3. **Flags** Bit[0]: Inverse Pattern Enable. Bit[1]: Exit on Uncorrectable Error: The test is stopped on the first uncorrectable error. Bit[2]: Error Count Threshold Programmed. Bit[3]: Reserved. Bits[7:4]: Addressing Mode (0h = Ascending, 1h = Descending, 2h = Algorithm Specific, 3h = Random). Bit[8]: Update Poison List on Uncorrectable Error. Other bits are reserved.
4. **Pattern Type** 00h = User provided. 01h = Vendor specific. 02h = PRBS. 03h = DPA[63:0] by eight. 04h = 55h. 05h = AAh. Other encodings are reserved.

Table 4 shows an example of Media Test Results Short Log that may be generated upon successful completion of the two tests. Note that one correctable error occurred during both tests.

Byte offset	Length	Description	Value
Common Header			
00h	1	Number of Tests Executed: 2	02h
01h	1	Version	01h
02h	1	Result (All the tests completed successfully)	00h
03h	0Dh	Reserved	0
Media Test Results for the first test			
10h	2	Test ID (Write, Read, and Compare pattern)	0002h
12h	8	Start Time	177F FA38 D345 570B h
1Ah	8	End Time	177F FA55 406B 1496 h
22h	1	Result (Completed with success)	00h
23h	1	Flags [1]	00h

Byte offset	Length	Description	Value
24h	4	Uncorrectable Error Count	0000 0000 h
28h	4	Correctable Error Count	0000 0001 h
2Ch	4	Reserved	0
Media Test Results for the second test			
30h	2	Test ID (MARCH)	0003h
32h	8	Start Time	177F FA55 7AF 9AE30 h
3Ah	8	End Time	177F FA9C D6CB CCC4h
42h	1	Result (Completed with success)	00h
43h	1	Flags (1)	00h
44h	4	Uncorrectable Error Count	0000 0000 h
48h	4	Correctable Error Count	0000 0001 h
4Ch	4	Reserved	0

Table 4 - Media Test Results Short Log

1. **Flags** Bit[0]: Error Signature List Overflow. Other bits are reserved.

Table 5 shows an example of Media Test Results Long Log that may be generated upon successful completion of the two tests. The Error Signature indicates that the correctable error was due to a single bit. The same error bit is detected in both tests.

Byte offset	Length	Description	Value
Common Header			
000h	1	Number of Tests Executed: 2	02h
001h	1	Version	01h
002h	0Eh	Reserved	0
Media Test Results for the first test			
010h	2	Test ID (Write, Read, and Compare pattern)	0002h
012h	8	Start Time	177F FA38 D345 570B h
01Ah	8	End Time	177F FA55 406B 1496 h
022h	1	Result (Completed with success)	00h
023h	1	Flags (1)	00h
024h	4	Uncorrectable Error Count	0000 0000 h
028h	4	Correctable Error Count	0000 0001 h
02Ch	4	Reserved	0
030h	8	Capacity Tested (128 GB)	0000 0000 0000 0200 h
038h	4	Number of Error Signatures	1
03Ch	4	Reserved	0
Error Signature 1 the first test			
040h	2	Iteration	01h
042h	8	Physical Address (2)	0000 0016 527A E340 h
04Ah	2	Validity Flags	007Fh
04Ch	1	Channel	1
04Dh	1	Rank	1
04Eh	3	Nibble Mask	00 02 00 h
051h	1	Bank Group	3
052h	1	Bank	0
053h	3	Row	00 B2 93 h
056h	2	Column	06 B0 h
058h	20h	Correction Mask	00 00 00 00 00 10 00 00 h 00 00 00 00 00 00 00 00 h 00 00 00 00 00 00 00 00 h 00 00 00 00 00 00 00 00 h
078h	10h	Component Identifier	

Byte offset	Length	Description	Value
088h	1	Sub-channel	0
089h	7	Reserved	0
Media Test Results for the second test			
090h	2	Test ID (MARCH)	0003h
092h	8	Start Time	177F FA55 7AF 9AE30 h
09Ah	8	End Time	177F FA9C D6CB CCC4h
0A2h	1	Result (Completed with success)	00h
0A3h	1	Flags [(1)]	00h
0A4h	4	Uncorrectable Error Count	0000 0000 h
0A8h	4	Correctable Error Count	0000 0001 h
0ACh	4	Reserved	0
0B0h	8	Capacity Tested (128 GB)	0000 0000 0000 0200 h
0B8h	4	Number of Error Signatures	1
0BCh	4	Reserved	0
Error Signature 1 for the second test			
0C0h … 10Fh		See from 040h to 08Fh	See from 040h to 08Fh

Table 5 - Media Test Results Long Log

1. **Flags** Bit[0]: Error Signature List Overflow. Other bits are reserved.
2. **Physical Address** Bit[0]: Volatile, Bits[2:1]: Error Type (00b = Uncorrectable, 01b = Correctable), Bit[3]: Inverse Pattern, Bits[5:4]: Reserved, Bits[63:6] DPA.

Table 6 shows Return Codes and Results that may be set in various cases after the completion of media test. It is assumed that the device was ready, and the media test was performed in background (Mailbox Status Register Return Code set to “Background Command Started”).

Description	*Background Command Status Register*	*Media Test Results Log*		
		Common Header	*Test 1 Media Test Results Entry*	*Test 2 Media Test Results Entry*
	Return Code	**Result**	**Result**	**Result**
No error	Success	All the tests completed successfully	Completed with success	Completed with success
At least one correctable error	Success	All the tests completed successfully	Completed with success	Completed with success
At least one uncorrectable error	Success	At least one test completed with a failure	Completed with failure	Completed with failure
Abort Background Operation command issued during the execution of the second test. No errors were detected during the first test.	Aborted	Test execution was interrupted by a Request Abort Background Operation command. All the tests that were complete, before the abort request was processed, ended successfully	Completed with success	Aborted by a Request Abort Background Operation command
Abort Background Operation command issued during the execution of the second test. At least one uncorrectable error was detected during the first test.	Aborted	Test execution was interrupted by a Request Abort Background Operation command. At least one test completed with failure before the abort request ended	Completed with failure	Aborted by a Request Abort Background Operation command
An internal error occurred during the execution of the first test	Internal Error	At least one test completed with a failure	Completed with failure	Completed with failure

Table 6 - Return Codes and Results for Media Test operations.

6 Conclusion

CXL 3.1 adds many optional RAS features that enable systems that contain CXL memory devices to uphold the RAS standard in datacenters today. The resiliency enabled by those features and the error logging enhancement provide a way for CXL devices to help maintain the availability and resiliency expected in at scale data centers. This white paper provided an overview of the new features as well as examples and configurations to help clarify the usage and intend of the RAS features that have been added.

With the ever-increasing memory footprints in data centers and the need for heterogeneous support, RAS will remain a top priority in the CXL Consortium in future intercepts. For more information about CXL please visit: https://www.computeexpresslink.org/.