An Overview of Reliability, Availability, and Serviceability (RAS) in Compute Express Link™ 2.0

Sudhanva Gurumurthi, Advanced Micro Devices, Inc.
Alexander J. Branover, Advanced Micro Devices, Inc.
Bryan Hornung, Micron Technology, Inc.
Vinny Michna, Hewlett Packard Enterprise Company
Mahesh Natu, Intel Corporation
Chris Petersen, Facebook, Inc.

January 30, 2021

Contents

1 Introduction and Whitepaper Overview

Reliability, Availability, and Serviceability (RAS) are important considerations in server design, deployment, and use in data centers. The Compute Express Link™ (CXL™) ecosystem comprises a multitude of component vendors (SoC, memory, storage, networking, etc.), system integrators, and end users. CXL enables a system vendor to populate memory devices, with differing internal architectures and media technologies in the same platform. One could also populate different coherent accelerators, each with a potentially different type of local memory and from different vendors. The CXL standard defines multiple RAS features, enabling all members within this rich, heterogenous ecosystem to innovate in their domains and still interoperate ultimately leading to RAS design points that meet the current and emerging end-user requirements.

This whitepaper provides an overview of the key RAS features of the CXL 2.0 specification published in November 2020. The organization of the rest of the paper is as follows. The paper begins with discussion of RAS in warehouse-scale computing infrastructures, a primer on key RAS concepts, and an overview of foundational RAS features in CXL that are common to both the 1.1 and 2.0 versions of the specification. The whitepaper then presents an overview of RAS enhancements that were introduced in CXL 2.0, recommendations on their usage, and some implementation examples. The paper then discusses the implications of CXL RAS for device vendors and interactions with system software. ***The content in this whitepaper is not to be considered normative. The CXL 2.0 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.

2 Data Center Trends and RAS

The explosive growth of internet content and the resulting data storage and computation requirements has resulted in the deployment of heterogeneous and complex solutions in very large-scale data centers. These warehouse-sized buildings are packed with server, storage, and network hardware and operate in a “lights out” fashion. The sheer scale of these deployments bring into sharp focus three aspects of RAS. First, even traditionally infrequent errors start to become visible as the aggregate volume of components and bits stored or transferred continues to increase. Any corner case events due to material degradation, environmental impacts, or manufacturing defects may affect the availability or performance of the systems. Secondly, error detection and remediation must be fully automated with minimal human interaction. As the number of repair events that require replacing hardware should be kept to an absolute minimum, there is the need to identify the failing component clearly and definitively, and minimize the time spent in repair. In addition, while error handling of persistent or permanent errors are easily automated, intermittent or transient errors may also occur frequently and have the potential to negatively affect application performance, uptime, and user experiences. As a result, the systems need to be designed to minimize the interference caused by RAS mechanisms and provide a deterministic behavior. All these effects add up to the need to build holistic, end-to-end solutions that leverage both hardware capabilities and software resiliency techniques.

3 RAS – Key Definitions

Reliability is the ability to provide correct service. One way to design a system to be reliable is to be able to detect and correct faults that may otherwise cause an error. An example of such a design is to use Error Correcting Codes (ECC) to detect a fault at a specific location in memory and correct it before the data stored in that location is consumed by a processor. *Reliability* is often expressed in terms of Mean Time Between Failures (MTBF), Mean Time To Failure (MTTF), or Failures in Time (FIT).

Availability is the ability to be ready to provide correct service, possibly under degraded capability or performance. An approach to attaining availability is to recover from an error so that the service is not interrupted due to the error. Using the memory example above, if an error is detected but is not correctable by ECC, one may offline (retire) the physical page that contains that specific memory address in the operating system so that that no future data is allocated to that location. In case recovery is not successful and the machine crashes and resets itself, then this would result in some duration of downtime, implying the system is unavailable during such downtime. One way to quantify availability is in terms of the "number of nines". For example, an *Availability* of 5 nines implies the system is up 99.999% of the time, i.e., approximately 5.25 minutes of downtime over one year of system operation.

Serviceability is the ability to diagnose and repair faults that may cause (or may have caused) an error. For example, memory repair techniques that allow a memory location that has an uncorrectable permanent fault to be physically repaired by replacing it with a spare would provide for serviceability. Serviceability may be "online", where the system continues to be available, or "offline", where the system is not available while being serviced.

In most state-of-the-art servers, it is important to provide reliability, availability, and serviceability, and hence the three are collectively referred to as "RAS". The specific mechanisms used to attain RAS may be a mixture of innovations in process technology, architecture, firmware, system software, libraries, and potentially application software. Each of these may be vendor and implementation specific. However, designing for RAS within the rich vendor ecosystem and multitude of use cases enabled by CXL requires standardization of certain canonical RAS features, a high degree of visibility into errors and other RAS-related events within a set of components interconnected by the CXL fabric, and the ability to log and communicate these events in a standardized manner to the host to enable RAS actions at the platform level and within operational flows at the data center level.

We will now describe how CXL 2.0 enables these server and data center requirements. The whitepaper first describes the RAS capabilities shared by CXL 1.1 and 2.0, highlighting changes when the specification evolved from the former to the latter. This is then followed by details of RAS enhancements new to the CXL 2.0 specification. A comparative summary of CXL 1.1 vs CXL 2.0 RAS is given in Table 1.

RAS Capability	CXL 1.1	CXL 2.0
PCIe-based RAS mechanisms for link and protocol errors	✓	✓
Data poisoning	✓	✓
Viral	✓	✓
Error injection for compliance testing	✓	✓
Function Level Reset (FLR) for CXL.io	✓	✓
CXL Reset for CXL.mem and CXL.cache		✓
Poison injection		✓
Support for viral propagation through switches		✓
Detailed memory error logging		✓
Hot-plug support		✓
Global Persistent Flush and Dirty Shutdown tracking		✓
Scan Media/Internal Poison List Retrieval		✓

Table 1 - A comparison of RAS capabilities in CXL 1.1 and newly introduced enhancements in CXL 2.0.

4 Foundational CXL RAS Attributes

4.1 Firmware-First and Operating System-First Error Handling

Firmware (FW)-First and Operating System (OS)-First are two methods for error handling in modern platforms and CXL accommodates both. The decision on which method to use is dependent on the system configuration involving devices, system firmware, and software layers involved for platform/device management. It is also feasible to implement a hybrid approach where certain sources of errors (e.g., link, device) are managed via FW-First and other sources are managed via OS-First. However, it is not advised to design both methods for same source of errors. Firmware notification (signaling) occurs by the device initiating an Error Firmware Notification (EFN), using Vendor Defined Messages (VDMs). For OS notification the device implements a standard Message Signaled Interrupt (MSI) method. Figure 1 illustrates the way the device notifies FW or OS about error event, using one of the abovementioned methods in CXL 1.1.

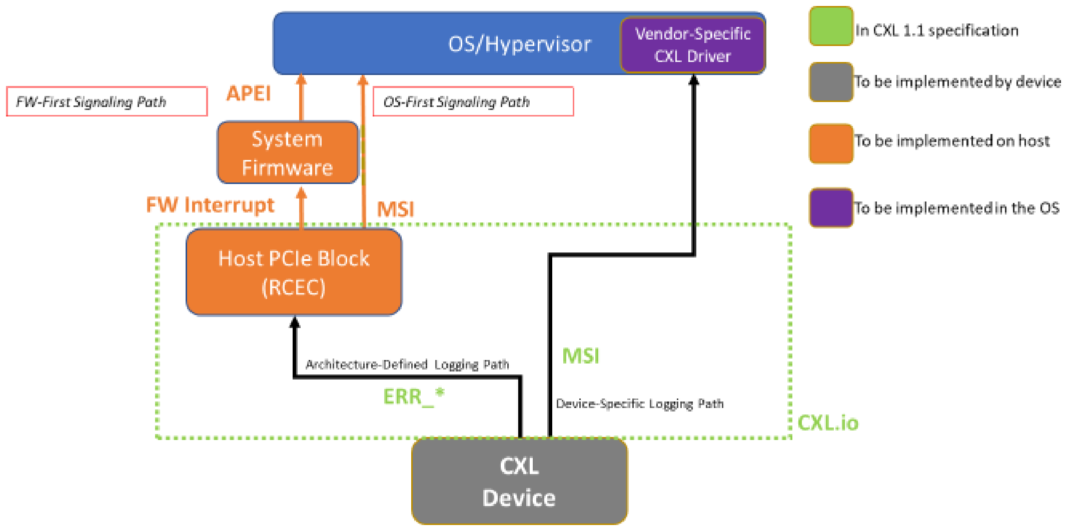

OS/Hypervisor
Vendor-Specific CXL Driver
FW-First Signaling Path
APEI
OS-First Signaling Path
System Firmware
FW Interrupt
MSI
Host PCIe Block (RCEC)
Architecture-Defined Logging Path
ERR_*
MSI
Device-Specific Logging Path
CXL.io
CXL Device
In CXL 1.1 specification
To be implemented by device
To be implemented on host
To be implemented in the OS

Figure 1 - Error logging and signaling in CXL 1.1.

In CXL 1.1, PCI Express® (PCIe®) AER provides the architected error logging path for CXL link and protocol errors. For errors signaled via this path, the host PCIe Block Root Complex Event Collector (RCEC) abstracts the choice of the specific method from the CXL device. When an error is signaled via AER, the RCEC in turn may generate a FW interrupt which invokes System Firmware in the FW-first method. System Firmware then calls APEI (ACPI Platform Error Interface) for error handling. Alternatively, in the OS-first method, the RCEC generates the MSI directly which invokes the CXL bus driver to handle the error.

While AER can be used for protocol and link errors, this signaling method has limitations for logging memory errors. First, AER makes it challenging to isolate where an error occurred. With AER, one only knows that an error occurred behind a CXL interface. Second, AER does not allow the address of a memory error to be logged. The lack of address information limits the ability of the host to get sufficient visibility into a memory error to offline a memory page – a widely used system-level RAS technique. Third, a CXL memory device may also have errors in its internal data path, errors related to media health, and errors that impact its ability to provide persistence. AER does not provide a distinct way to report errors attributable to these events to the host. Therefore, *AER (and hence FW-first error handling) is not recommended for memory error logging in CXL 1.1*. AER with the FW-First method can still be used for CXL.io devices.

The CXL device can also use direct MSI messaging to the host to call the CXL vendor-specific device driver for the error log processing, as shown by the arrow on the righthand side path in the figure. Since most Type 1 and Type 2 devices are expected to have a vendor-specific driver and this driver may be involved in managing the device including the HDM, this mechanism is suitable for these devices. Type 3 devices, on the other hand, are not likely to have a vendor-specific driver and the HDM produced by these devices is often directly managed by OS. Therefore, a vendor agnostic error logging and signaling mechanism is required for Type 3 devices.

CXL 2.0 allows the device to initiate standardized OS notifications and to support platform level RAS features and actions using a generic OS stack, without dependency on the vendor-specific device driver. At the same time, both OS/Kernel RAS management of the device can co-exist with the vendor specific device driver if the device allocates separate and non-overlapping logging of the error events. CXL 2.0 also allows capturing a much richer information about errors in memory devices without the need for a vendor-specific driver, as will be described in Section 5.1. *It is strongly recommended that Type 3 devices follow the CXL 2.0 specification for logging and signaling memory errors.*

4.2 Leveraging PCIe RAS Mechanisms

The CXL architecture relies on the CXL.io protocol for device discovery, configuration, and interrupts. CXL link and protocol error reporting follows this scheme and leverages CXL.io (PCIe) error logging and signaling. Examples of CXL using PCIe error mechanisms include the following:

1. CXL does not define new error messages and leverages PCIe ERR_* and MSI messages.
2. CXL link errors are logged and signaled just like PCIe link errors.
3. CXL.io protocol errors are logged and signaled just like PCIe protocol errors.
4. CXL.cache and CXL.mem protocol errors are logged in CXL-specific registers, the format of which closely resembles PCIe AER Extended Capability.
5. For signaling purposes CXL.cache and CXL.mem errors are overlaid on top of PCIe AER. As such, they are logged in the associated PCIe Function either as Correctable Internal Errors (CIE) or Uncorrectable Internal Errors (UIE) based on error severity and reported to the host via standard ERR_* messages.

Mapping CXL errors onto PCIe AER ensures that existing PCIe software can handle these new error conditions in a functionally correct way (e.g. stop the system on an uncorrectable CXL.cache and CXL.mem protocol error). This is a significant benefit and provides the end users the flexibility to independently update the hardware and software components. The downside of this approach is the loss of flexibility and granularity in the types of errors that can be reported. For example, all uncorrectable CXL.cache and CXL.mem errors are either signaled as ERR_NONFATAL or ERR_FATAL depending on how software configures the “Uncorrected Internal Error Severity” bit in PCIe AER Capability.

CXL.io also supports PCIe Enhanced Downstream Port Containment (eDPC) to disable links upon detecting uncorrectable errors to contain and enable recovery from such errors. *Enabling eDPC is not recommended in most CXL 2.0 systems because eDPC containment flow brings the link down, disrupting CXL.cache and CXL.mem traffic which can lead to host timeouts.*

4.3 Reset Mechanisms for Recovering from Device Hangs

When a CXL device has stopped responding, it may be necessary for the host to quiesce the device and reset it to attempt recovery. The PCIe standard provides the ability to do this via a Function Level Reset (FLR). CXL inherits FLR from the PCIe standard and supports FLR on CXL.io. However, FLR has no effect on CXL.cache and CXL.mem. To support recovery from hangs of these types of CXL devices, CXL 2.0 introduces a new reset mechanism called CXL Reset. A CXL Reset has the same effect as FLR on CXL.io. In addition, CXL Reset also reinitializes CXL.cache and CXL.mem domains and is also capable of reinitializing HDM contents. All CXL 2.0 SLDs that participate on CXL.cache or CXL.mem are required to support either FLR or CXL Reset. The Multi-Logical Device (MLD) are required to support CXL Reset.

As with FLR, system software is responsible for orchestrating a CXL Reset and offline any associated HDM ranges. Once a CXL Reset is complete, all CXL Functions on the impacted device must be re-initialized prior to use. Details of the CXL Reset flow at the device level and required system software actions to perform a CXL Reset and subsequently re-initialize the device are given in the CXL 2.0 specification.

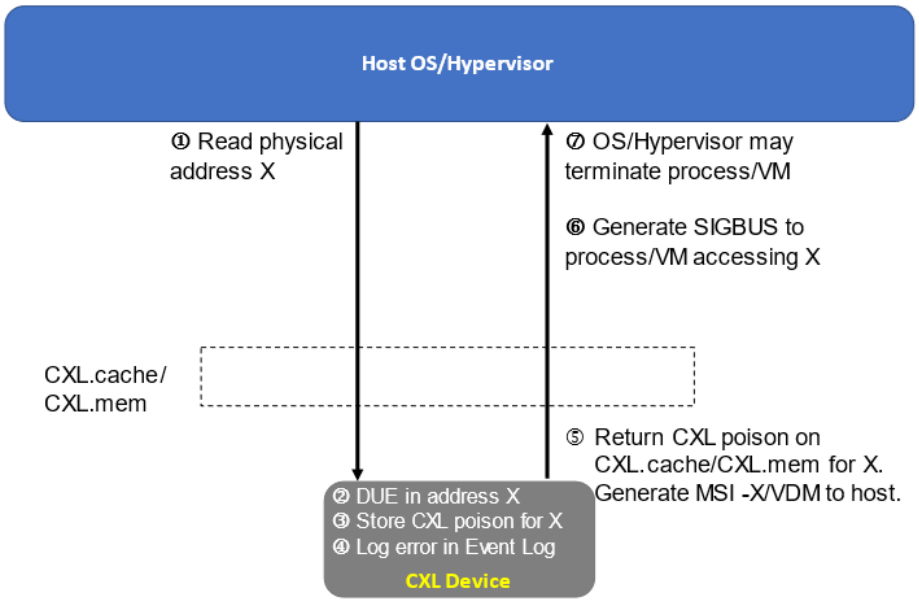
If neither FLR nor CXL Reset is available or their use does not bring the device out of an error state, software may use a stronger form of reset, i.e., secondary bus reset. Secondary bus reset reinitializes the CXL link and the device. Prior to issuing a secondary bus reset, system software is responsible for quiescing the device and offlining any associated HDM ranges.

4.4 Standardized Communication Mechanisms for Error Containment

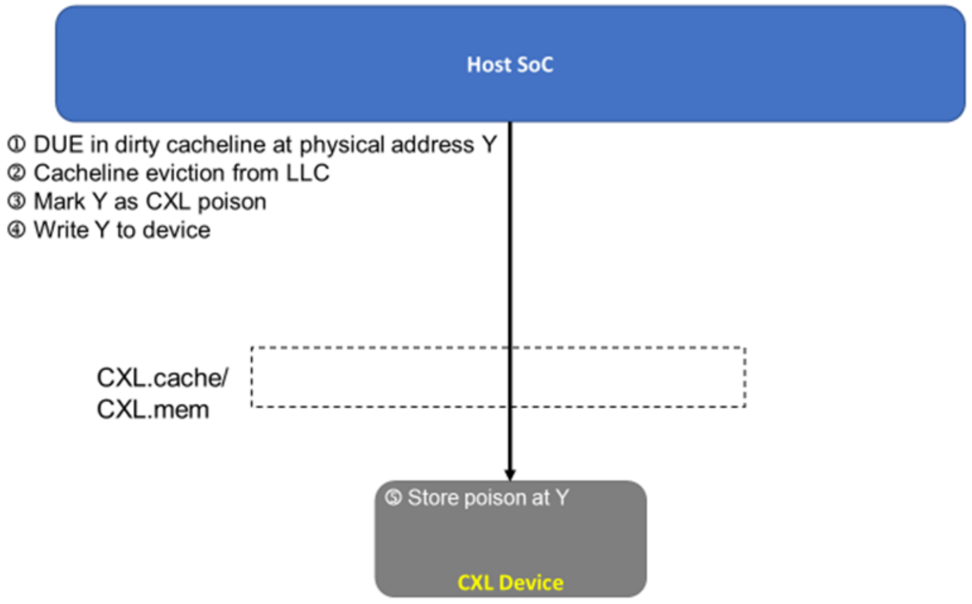
CXL enables interconnecting coherent accelerators and memory on the platform. Since these different platform components may be from different vendors, it is important to have standard ways of informing whether any data returned by a CXL device may be bad or suspect. This is critical so that the consumer of the data can take appropriate containment actions upon its receipt in a platform-specific manner.

CXL provides two mechanisms for a device, whether it be a compute or memory device, to communicate data corruption or a broader failure event. These mechanisms are *poison* and *viral*. All CXL devices are required to support poisoning and keeping track of poisoned locations, and all CXL links and devices are expected to support viral. Both CXL hosts and devices can use these two containment mechanisms. We now explain these two containment mechanisms, with examples.

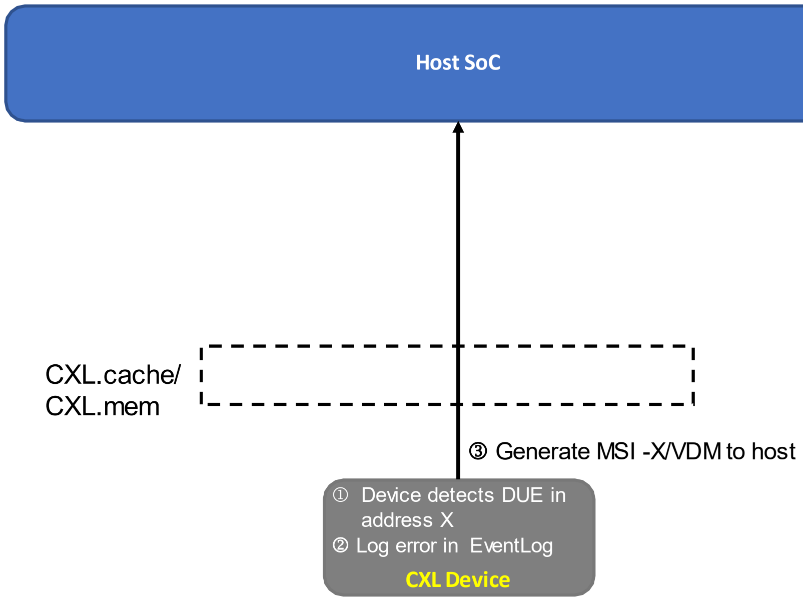
4.4.1 Data Poisoning


Host OS/Hypervisor
① Read physical address X
⑦ OS/Hypervisor may terminate process/VM
⑥ Generate SIGBUS to process/VM accessing X
CXL.cache/ CXL.mem
⑤ Return CXL poison on CXL.cache/CXL.mem for X. Generate MSI -X/VDM to host.
② DUE in address X
③ Store CXL poison for X
④ Log error in Event Log
CXL Device

(a) Poison generated by a CXL device on a host read and consumed by the host.



(b) Poison generated by the host and sent to a CXL device.


Host SoC
CXL.cache/
CXL.mem
③ Generate MSI -X/VDM to host
① Device detects DUE in address X
② Log error in EventLog
CXL Device

(c) Device detects and logs poison independent of a host access.

Figure 2 - Examples of data poisoning between a CXL host and device. X and Y are physical addresses. The device translates a Host Physical Address (HPA) to the corresponding Device Physical Address (DPA) and vice-versa. DUE = Detected but Uncorrectable Error. LLC = Last Level Cache.

Poison is used to indicate that a specific data returned by the device has an error that could not be corrected by the device. Upon receipt of poison, system software may be able to contain the error to the process/virtual machine (VM) that accessed the data, which can enhance availability. Three examples of data poisoning are given in Figure 2. Figure 2(a) shows a case where a CXL device generates and stores poison for a specific physical address as a result of a Detected but Uncorrectable Error (DUE) in the device when servicing a read request to that data by the host. In this case, poison is returned to the host for that read and this error is also logged by the device in the Event Log mailbox (Section 5.2). For a Persistent Memory (PMEM) device, the poisoned location and the poison source are also added to the internal Poison List maintained by the device. The poisoned data read from the CXL device is stored in the host's cache where it may be consumed by a processor. When the OS/hypervisor detects that the address accessed by the process/VM is poisoned, it generates a Bus Error (SIGBUS) exception, which typically leads to termination of the process that attempted accessing that data. Figure 2(b) shows a case where the host generates poison. The specific example in the figure illustrates the case where there was a DUE in a dirty cache line in the Last Level Cache (LLC) of a processor and is being evicted and written to the CXL device. In this case, the host processor poisons the cache line and sends it to the device. The CXL device stores the poison at the corresponding address. For a PMEM device, the poisoned location and the source are also added to its internal Poison List. A CXL device may also encounter a DUE at a specific address independent of a host access, generate and log poison, and inform the host. This scenario is illustrated in Figure *2*(c).

CXL 2.0 defines **`Get Poison List`** and **`Scan Media`** commands to retrieve poison information. **`Get Poison List`** provides a complete list of poisoned locations on the device with low latency. However, if the Poison List has overflowed or if the device indicates that a media scan is necessary to retrieve the complete list of poisoned locations, **`Scan Media`** is to be used. **`Scan Media`** is required for persistent memory regions where poison will not be cleared via writing the media during the reset flow. More details on this are presented in Section 5.4.3.

4.4.2 Viral

Viral is used to indicate a more severe error condition on the device where an uncorrectable error cannot be attributed to a specific address or range of addresses. All data returned by the device after it has communicated that it has gone viral is considered suspect. The key goal of viral is to avoid data corruption by preventing any bad data from being committed to persistent memory, regardless of whether that memory is local to the device or connected via an external interface. (NOTE: The CXL specification recommends that poison not be signaled when a device is in the viral state). The CXL specification does not define a mechanism to recover the system from viral condition without a system reset and, in most systems, a system reset may be required.

Viral can be communicated in both directions between a host and a device in response to an uncorrected fatal error. Figure 3 shows an example of CXL device encountering an uncorrected fatal error condition and triggering viral notification to the host. In Figure 4, the host detects an uncorrectable fatal error and sends out viral notification to all connected CXL devices.

Viral notification is not precise in terms of the affected data packets. It is designed to propagate across CXL faster than any transaction that could have been affected by the error condition that triggered viral. When a device receives viral or detects an error that triggers viral, the device must stop all writes to persistent memory ranges to ensure containment. A MLD must take care to contain the viral condition to only the logical devices affected by the error. Details on viral for MLD is covered in Section 5.3.2. A PMEM device receiving viral notification should ensure that all contents buffered within the device are flushed to persistent media. If the integrity of the PMEM data is compromised, a "dirty shutdown" is considered to have happened (please see Section 5.4.2 for details).

When the host receives a viral notification, it takes error containment actions to avoid data corruption in other persistent media including CXL devices attached to other root ports. The specific actions taken by a host in response to viral will be implementation specific.

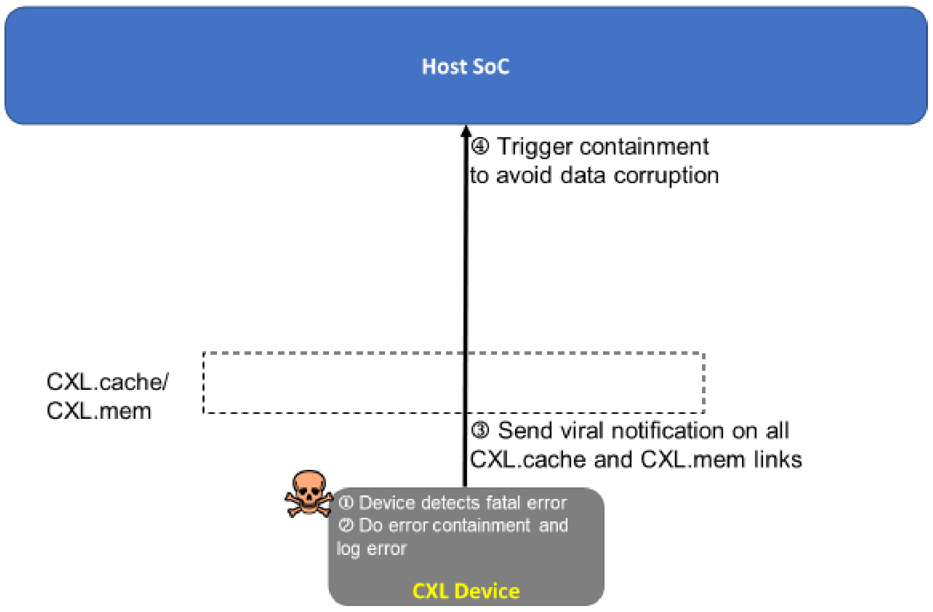

Host SoC
④ Trigger containment to avoid data corruption
CXL.cache/ CXL.mem
③ Send viral notification on all CXL.cache and CXL.mem links
① Device detects fatal error
② Do error containment and log error
CXL Device

Figure 3 - CXL Device to Host Viral notification

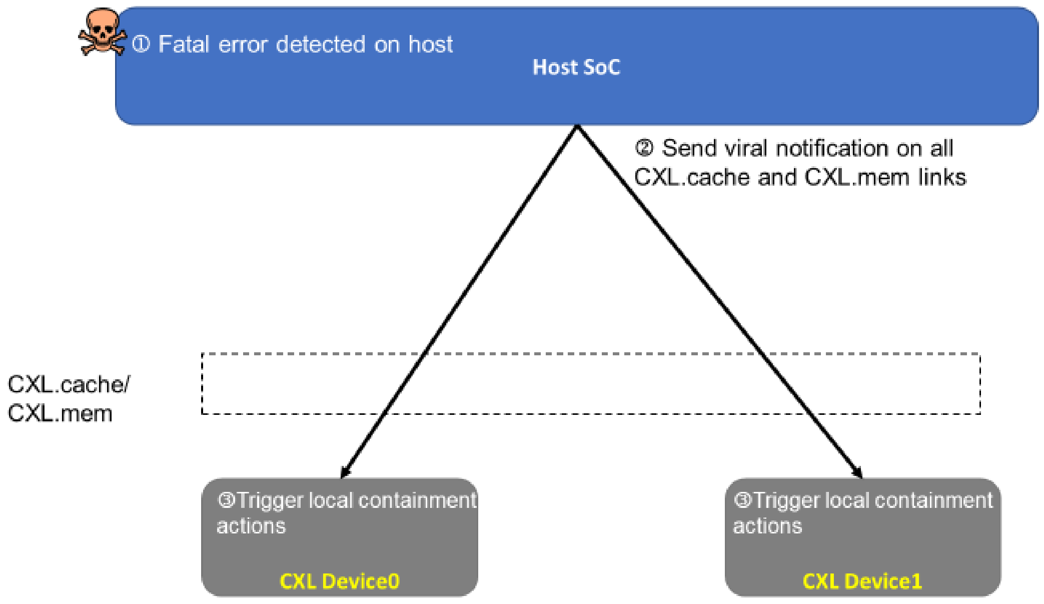

① Fatal error detected on host
Host SoC
② Send viral notification on all CXL.cache and CXL.mem links
CXL.cache/ CXL.mem
③Trigger local containment actions
CXL Device0
③Trigger local containment actions
CXL Device1

Figure 4 - Host to Device Viral propagation

5 CXL 2.0 RAS Enhancements

5.1 CXL Memory Error and Event Logging and Signaling

Error signaling and logging are important requirements for RAS at the CXL link, host, device, and system level to ensure that any errors that occur within a system are diagnosed and resolved in a time-efficient manner. To that end, the specification defines a common set of registers, logs, and messages to record and report protocol, link, and device errors in a vendor-independent fashion. These common specified definitions for error reporting enable system software/firmware to detect and correct errors without requiring knowledge of each vendor-specific technology and implementation methods.

At the link layer, in addition to the standard PCIe error reporting mechanisms (e.g., AER messages) that are reported over the CXL.io interface, the specification defines a standard set of RAS specific registers that can be accessed from the memory space of the device (as part of the Component Registers) to quickly determine the source of the error. CXL-aware software can extract the details of link and protocol errors from these registers to provide additional insight into source/cause of the error.

At the device level, CXL 2.0 defines the mechanism to access the device error logs through a standardized register interface. Of particular concern, memory errors can occur in a CXL device independently of any host issued requests, and as a result it is important to log data related to these errors to enable the use of host or platform-level RAS features, such as page offlining (also known as page retirement), without dependence on a vendor-specific driver. Such error logging is required for Type 3 CXL devices that provide a direct interface to the host (e.g., DRAM), as well as devices that have an abstracted interface to manage the specifics of their memory technology. By supporting a common error logging mechanism, CXL 2.0 can support error logging without vendor driver dependence, while continuing to support the co-existence of both host- and device-level RAS features and logging. The specification defines multiple error logging queues for CXL devices, with each queue being based on the severity of the errors. Software/firmware can monitor these different error queues and take actions, as necessary.

At the system level, CXL 2.0 enables device errors to be signaled to the host through either software or firmware-based interrupts. This allows system designers to implement the detection and processing of errors with either a firmware first, or software/OS first methodology. For firmware first implementations, the specification defines an Error Firmware Notification (EFN) VDM that is sent as an interrupt to the host over CXL.io. For software/OS first error processing, events are signaled through the same MSI/MSI-X mechanisms that are currently used in PCIe.

5.2 Enhanced Memory Event Logging and Handling

A CXL system comprises of multiple memory controllers. In addition to the host memory controllers, each Type 3 device carries one or more local memory controllers. The CXL protocol abstracts away the memory controller in a Type 3 device. This architectural choice simplifies the interface between the host and memory, allows memory vendors to innovate on their media technology and memory device architecture, and provides flexibility to the platform in using a mixture of memory device types (e.g., DRAM + NVRAM). However, as discussed earlier, many platforms and operating systems require a high degree of visibility into memory errors and RAS-related events within these devices. For example, DRAM page retirement requires the address of an error to be known to system software to offline a page. Many data centers also collect memory error related telemetry information to attain high resilience at scale. These RAS algorithms may employ Predictive Fault Analysis (PFA) techniques based on the telemetry data. Achieving these capabilities requires detailed logging and signaling of errors and other key events by CXL memory devices and allowing system software to access them in a standardized manner without the need for a vendor-specific driver. CXL 2.0 facilitates these RAS use cases via the mechanisms shown in Figure 5.

CXL 2.0 defines a set of Event Records that capture a wide range of error types and events that accommodate the diversity of memory media types, device architectures, and failure domains within memory devices. CXL memory device controllers may also use sophisticated algorithms to manage the reliability of the local media. Such algorithms may be able to predict when an error may occur that may cause data loss. CXL event logging allows CXL memory devices to inform the host of such predictions in a standardized way. For instance, the host could use such an event notification to proactively backup data stored on the device.

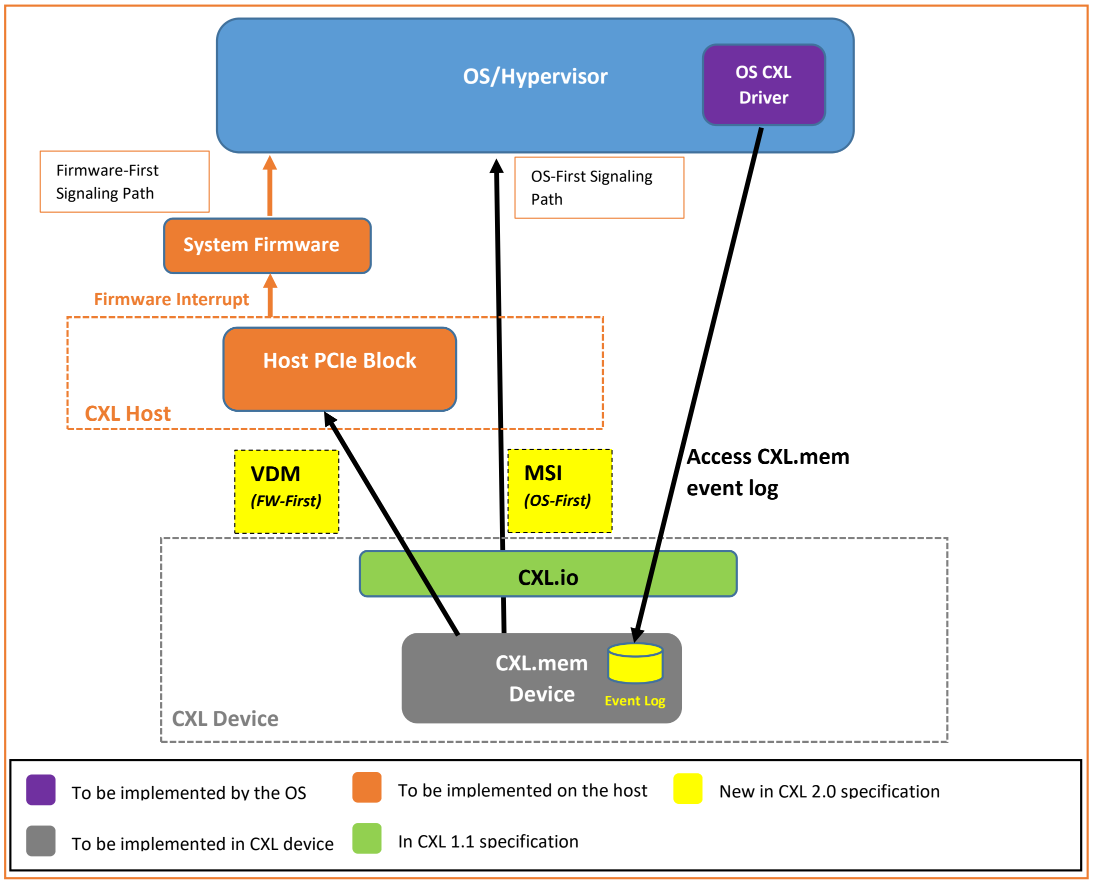


Figure 5 - Memory error and event logging in CXL 2.0.

Memory devices log these events in an Event Log mailbox on the device and can signal the host via an MSI/MSI-X or an EFN VDM. The CXL event logging enhancements provide a common view of memory errors and events across all CXL platforms and support both OS-first and FW-first error handling.

5.3 Enhancements to Viral in CXL 2.0

5.3.1 Viral Propagation Through Switches

Errors detected by upstream and downstream ports of a CXL switch are escalated to the root port of the CXL virtual hierarchy using standard PCIe error reporting mechanisms. In the case where a CXL switch receives a viral indication from the host or device, the switch will propagate the viral indication on all switch ports that are connected within the same virtual hierarchy.

5.3.2 SLD vs. MLD Devices

A pooled Type 3 memory device can partition its resources into Logical Devices (LD), where each LD appears as a Type 3 device. CXL 2.0 allows up to 16 LDs per MLD. A Virtual CXL Switch (VCS) propagates the Viral indication to all the LDs associated with the VCS by setting the Viral bit in the Viral LD-ID vector for the LDs in that VCS.

An LD within the MLD component that has entered the viral state sets the Viral bit in the CXL.mem traffic with the mask set to identify all the LD-IDs associated with all the affected VCSs. The indication from each LD-ID propagates the viral state to all associated VCSs that have viral containment enabled. An MLD device running into Viral condition sets Viral status bit and takes steps to contain the error within the device (specifically logical devices impacted by the viral condition). In addition, the device communicates the viral condition to all the hosts associated with the Virtual Hierarchy.

Links without LD-ID support (referred to as SLD – Single Logical Device) treat the LD-ID vector as Reserved. For an MLD, the encoding of all zeros indicates that all LD-IDs are in viral and is equivalent to an encoding of all ones.

5.4 RAS for Persistent Memory Devices

5.4.1 Global Persistent Flush (GPF)

When a power failure occurs on a host connected to a device with a persistent HDM range, it is important to flush all stores in memory to a persistence domain. This is important to ensure that writes to memory that are held in volatile storage (e.g., internal processor or device caches and buffers) are not lost, as they may already be deemed to be persistent from the viewpoint of software. GPF is a hardware-based mechanism to accomplish this flush-to-persistent media and all hosts, switches, and PMEM devices that comply with CXL 2.0 are required to support it.

The GPF sequence is managed using a sequence of VDM messages and consists of two phases, the first for performing a cache flush and the second for devices to flush their local write buffers to the persistent domain. The GPF payload indicates whether a cache flush is required for the first phase, as not all systems are required to supply energy to flush processor caches and CXL.cache devices. On a successful GPF, the device clears the Shutdown State variable and preserves it in a non-volatile region.

Error conditions may arise during the GPF flow related to flushes to the persistent domain or to internal device caches/buffers. For example, a second GPF VDM message may appear at the device before a previous one has been completed. The device may choose to log this event as a warning in a Vendor Specific Event Record. Host and device actions on conditions such as these are implementation specific. Another potential error condition is that a GPF itself fails, which is discussed next.

5.4.2 Dirty Shutdown Count (DSC)

If GPF fails, a “dirty shutdown” is said to have occurred. The Dirty Shutdown Count (DSC) is a monotonically increasing counter incremented whenever the device fails to save and/or to flush data to the persistent media or unable to determine if data loss may have occurred. The media health status keeps track of the Dirty Shutdown events by using Dirty Shutdown Count (DSC). The count must remain persistent. Two parameters – Shutdown State and DSC are used to report the shutdown status. The DSC can be retrieved by the host via the **`Get Health Info`** mailbox command.

On reset events, if a device detects an internal failure during any write to persistent memory and this failure compromises the data integrity of persistent media, the Shutdown State is set to Dirty. The Dirty state needs to be preserved through the rest of the reset flow.

On wake-up from reset, the device checks the Shutdown State. If the Shutdown State is Dirty, the DSC is incremented, and the Shutdown State is cleared. Host software can use the DSC to detect dirty shutdowns and react based on the PMEM programming model used on the system.

In addition to the DSC, the Memory Module Event Record contains fields to log information about detected and potential persistence failures. This information can provide additional diagnostic information to the host about the failure and potentially aid recovery. Details of how a device may log this information and how a host can leverage them are implementation specific.

5.4.3 Scan Media and Internal Poison List Retrieval

The **`Get Poison List`** and **`Scan Media`** commands and Poison List internal to a CXL device facilitate discovering faults in persistent memory. A host access to a memory location with such a fault may lead to a DUE. A benefit of communicating information about such fault locations to the host is that it allows system software to avoid accessing these locations.

A typical use case is for the host to invoke **`Get Poison List`**, which provides a complete list of poisoned locations on the device with low latency. However, if the Poison List has overflowed or if the device indicates that a media scan is necessary to retrieve the complete list of poisoned locations, **`Scan Media`** is to be used.

Invoking the **`Scan Media`** command on a device triggers a potentially long background scan of its persistent memory. This obviates the need of the host to explicitly scan the media by sending CXL accesses or to even request device to do a media scan.

Any discoverable uncorrectable errors need to be presented across power cycles which avoids any need for the host to explicitly ask the device for the media scan. At the same time, the host still can trigger a scan if desired through the **`Scan Media`** command. A device can notify the host about an outcome of its internal scan by generating MSI or VDM notifications. Once notified the Host can take a proactive set of actions to address the poisoned media ranges. The locations that are found to have uncorrectable errors are poisoned and added to the Poison List.

Anytime the device adds new locations to its internal poison list, the device can notify the Host that the list has been updated by generating MSI or VDM notifications. The Host can also clear a poisoned location using the **`Clear Poison`** command, which removes that specific physical address from the Poison List of the device atomically.

5.5 Hot-Plug

A key addition to the CXL 2.0 specification is the ability to hot-add and perform a software-managed hot-remove of hardware (devices and/or switches). Hot-plug support benefits RAS in two ways. The ability to hot add or remove hardware allows other components and agents in the system to potentially continue running, which benefits availability. Hot-plug can also be used in response to a hardware failure of a field-replaceable unit, which enhances serviceability. Note that surprise removal of hardware is not supported and will result in unpredictable behavior.

5.6 Error Injection

Error injection in CXL serves three purposes:

(1) A means for testing the system response to an error, which is useful for validation purposes and for software/firmware development.
(2) Compliance testing for adherence to various aspects of the CXL specification.
(3) Hardware-specific testing. For example, such testing can aid in device driver development. Hardware-specific testing is outside the scope of the CXL specification and is left as a vendor-specific implementation choice.

The CXL 1.1 specification includes recommendations for specific types of error injection for RAS purposes and requirements for compliance testing. CXL 2.0 expands the compliance testing uses of error injection and introduces poison injection for CXL.mem.

5.7 RAS and Security Interactions

When CXL.cache and CXL.mem Integrity and Data Encryption (IDE) is enabled, the components are required to verify that all received protocol flits are accompanied by valid integrity (MAC). If the integrity check fails or MAC slot is not received in time, the receiver is required to treat this as an uncorrectable error and drop all subsequent protocol flits. Even though this strong containment action is necessary from security perspective, the dropped flits can result in fatal timeouts on both ends of the link. MAC check is always performed after link CRC check and any retries. As a result, it is expected that transient bit flips will be corrected at the link level and a wrong MAC is, in most cases, caused by an attack on the link.

CXL.cache/CXL.mem IDE can either be operated in Skid mode or Containment mode. The containment mode ensures that any data that may be suspect from security standpoint is not forwarded to the consumer. However, the stronger containment comes with a performance cost, in terms of both latency and bandwidth. The performance cost of skid mode is very low, but the up to 128 flits worth of data is forwarded to the consumer before the MAC is verified.

Certain segments require that all data paths inside a component provide adequate data protection. For a CXL.cache/CXL.mem IDE capable component, these data paths include the IDE encryption/decryption pipeline stages. Due to the properties of AES, a single-bit error inside the AES engine may result in an avalanche of errors and up to 50% output bits may be flipped. CXL architecture facilitates this usage model by way of encrypted PCRC. CXL.cache/CXL.mem transmitter calculates CRC over the plaintext and appends the encrypted version of this CRC to the ciphertext before computing the MAC. The encrypted CRC itself is not sent over the link because the receiver is able to recalculate it after decryption phase, append it to ciphertext and reverify the MAC to ensure that the data did not get corrupted during the decryption process.

6 Implications of CXL RAS to Device Vendors

This section categorizes some examples of common device errors and provides examples of how they can be logged and possibly handled by a system. The errors are categorized using the Error Reporting and Nomenclature Guidelines found in the CXL 2.0 specification and it is assumed that the device has implemented the Memory Error Logging and Signaling Enhancements (Mailbox) capability described in Section 5.2. A block diagram is shown in Figure 6 as a simple device model. A key part of that model is a device management controller that collects event logs and can communicate with the host software.

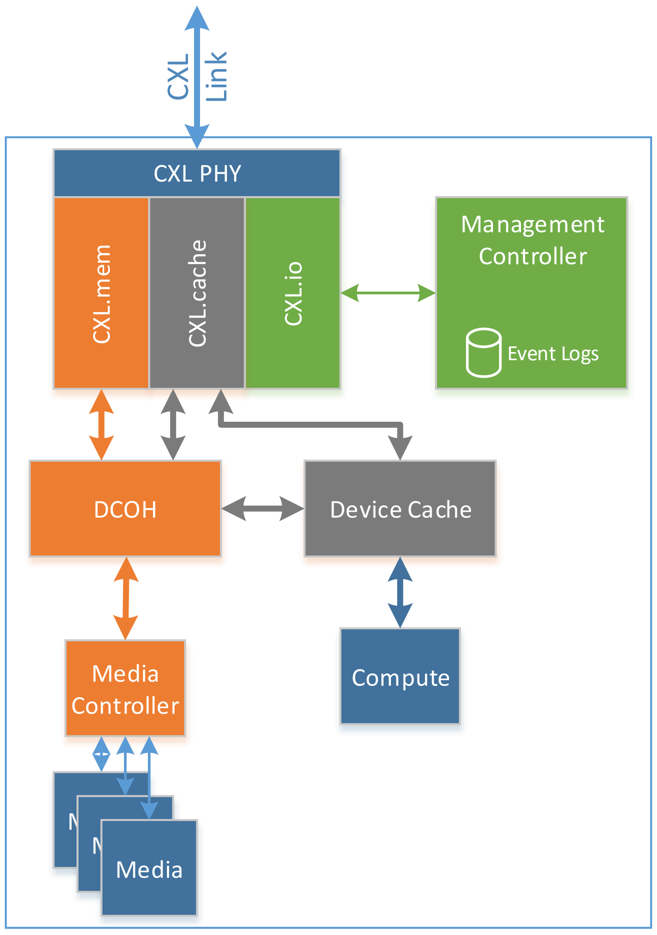


Figure 6 - Simplified Type 2/Type 3 Device Block Diagram. DCOH is the Device Coherency agent, defined in the CXL 2.0 specification.

6.1 Corrected Errors

Corrected errors are those that are detected and corrected by hardware. These errors do not generally require system software intervention for the continued correct operation of the system. System software may be invoked to prevent the error condition from growing into an uncorrected error. Examples of this type of error in a CXL device include the following:

- Correctable error in media
- Correctable data error in the device cache. This information may be used by the device as an indicator of device health and not be provided separately to the host
- Correctable error in the device data path (This may just be used by the device as an indicator of device health)
- Link CRC error
- Receiver errors

To further illustrate how corrected errors may be handled in a CXL based system, this section will use a corrected media error as an example. The device may want to expose these to the system so that an OS can offline a memory page where too many corrected errors are occurring, or an administrator may want to schedule maintenance to replace a device. To support those use cases, a memory expansion device will record corrected errors in the Informational Event Log. The event logs can be polled using the **`Get Event Records`** mailbox command or the device can be configured to interrupt the host when the event log transitions to the not empty state. Additionally, the **`Alert Configuration`** command can be used to set the threshold at which correctable errors generate an event. The host may then access the event log to determine if a page should no longer be used. If the corrected errors are not limited to a small set of specific addresses maintenance can be scheduled for the device. The device should maintain separate correctable memory error counters for volatile and persistent ranges which are reported via the **`Get Health Info`** mailbox command.

NOTES (The following are guidelines – implementations will be vendor-specific):

- A subset of the corrected errors in the media may be masked from reporting. One example for this is due to a high media-level fault rate. Filtering such errors can reduce the overheads of error logging on the device, reduce the likelihood of log overflows, and help manage interrupt storms at the host. The extent to which corrected errors may be filtered will be implementation specific. *It is recommended that the filtering threshold be guided based on the characteristics of the underlying media technology, the specifics of the platform RAS architecture, and the system-level RAS goals of the end-users of the devices.*
- For certain memory devices, there may be an additional layer of mapping between a DPA and the media physical address. For example, if the memory device employs wear-leveling to manage media endurance, writes to a single DPA may map to different locations on the media. In such cases, page offlining may be best carried out by the media controller inside the memory device and a software-driven page retirement approach may be counterproductive. Whether a memory device supports such internal page offlining is a vendor-specific implementation choice.

6.2 Uncorrected Recoverable Errors

Uncorrected recoverable errors are uncorrected errors detected by hardware and can recover from with minimal or no software intervention. Although data can be lost with these errors, that data is not required for the continued operation of the system. Examples of this type of error in a CXL device include:

- Uncorrectable error in device cache during dirty cast-out, poison written to memory, not consumed by a process
- Uncorrectable error in device data path in data fields, not consumed by a process

As an example, an uncorrectable error in the media would be logged in a Warning Event Log by a memory expansion device and an interrupt would be immediately sent to the host if enabled. If the address is part of a persistent memory region, the poison list is updated with the failing DPA. Upon reading the event log the host would immediately attempt to stop using the page that contains the DPA from the event log. If the host determines that the page is available for continued use it should write to the failing DPA to ensure poison is cleared or utilize the **`Clear Poison`** mailbox command.

6.3 Uncorrected Non-Fatal Errors

Uncorrected non-fatal errors are uncorrected errors detected by hardware that do require software intervention. As with uncorrected recoverable errors, the device may continue to operate correctly except for the data identified as affected (e.g., poisoned locations). Unlike uncorrected fatal errors, the device may have failed but the data lost does not impact the operating system. Data can be lost with these errors and affected processes must be terminated and re-started. Examples of this type of error in a CXL device include the following:

- Uncorrectable error in the media, return poison with read response, consumed by a process
- Uncorrectable error in device cache during dirty cast-out, poison written to memory, consumed by a process
- Uncorrectable tag error in the device cache, all snoops return poison data even if the host knows this is shared
- Uncorrectable error in device data path in data fields, consumed by a process

- Completion Time Out

As with the uncorrected recoverable case, the example uncorrectable error in the media would be logged in the Warning Event Log because the difference between recoverable and non-fatal is that the recoverable error resulted in poison consumed by a host processor. Although the memory expansion device signaled the event using an interrupt, the logs will be read by the host in the process of recovering from the poison consumption.

6.4 Uncorrected Fatal Errors

Uncorrected fatal errors are uncorrected errors detected by hardware that pose a containment risk. Unlike uncorrected and recoverable non-fatal errors, the data lost impacts too much of the system to allow continued operation of the system or could lead to writes of corrupted data to persistent storage. The system must be reset and restarted, if possible, to enable continued operation. Because this error affects the entire device, a persistent memory device is considered to have experienced a dirty shut-down. Examples of this type of error in a CXL device include the following:

- Uncorrectable error in device control path (non-data fields)
- Device may assert viral for containment if enabled by the host.

An uncorrected error in a device control path will be logged as a fatal event whether viral signaling is enabled or not. The device will signal the event via an AER message, but the CXL viral signal, if enabled, will reach the host before the AER message. A viral condition in the host is fatal and system firmware will use the device Fatal Event Log to determine the source of the fatal error. In a device that supports MLDs this error should be contained to the smallest subset of logical devices possible.

7 System Software Support for CXL RAS Features

As discussed earlier, in CXL 1.1, a CXL device notifies the host of an error either via AER, through the PCIe block RCEC, which then generates a firmware interrupt or an MSI, or via an MSI to a vendor-specific device driver. In the case of FW-first error handling, system firmware would use the ACPI Platform Error Interface (APEI) and Hardware Error Source Table (HEST) to notify the OS of the device error. In CXL 2.0, a memory device can directly generate an MSI to host without a vendor-specific driver if OS-first error handling is used. This requires that the OS be CXL 2.0 aware. The Operating Systems Capabilities (`_OSC`) method allows an OS to communicate to the platform whether it supports CXL 2.0 and negotiate with the system firmware about ownership of memory device errors. If an OS supports only CXL 1.1, device errors can be communicated to the host only via PCIe AER rather than via the Event Log mailbox.

8 Conclusion

CXL 2.0 enables the flexible composition of compute, memory, storage, and networking while enabling resilience at scale. This white paper has provided an overview of key RAS capabilities of the CXL 2.0 specification and provided recommendations on their usage. RAS continues to be a topic of interest in various CXL working groups for future intercepts of the CXL standard with efforts underway in defining new mechanisms that build on the concepts and features described in this paper. For more information about CXL, please visit https://www.computeexpresslink.org/.