EXPLORING RAS ACTIONS

A Comprehensive Review and Key Learnings from Internship

ANSHUL GUPTA

Table of Contents

1. OVERVIEW

This paper provides an overview of my internship experience as a Platform Application Engineer (PAE) working with Intel® Xeon® processors, specifically focusing on their Reliability, Availability, and Serviceability (RAS) features.

The topics covered in this paper include an introduction to Intel® Xeon® processor RAS, as well as detailed discussions on System, Memory, and PCIe RAS features that I worked on during my internship.

Additionally, the paper highlights the support I provided to the team as part of my role as a PAE. It also outlines my understanding of server platforms, RAS functionality, and the specific RAS features I encountered, along with brief descriptions of the supported features.

2. INTRODUCTION

The term Reliability, Availability, and Serviceability (RAS) is a comprehensive set of features designed to ensure that the systems can operate reliably, remain available for use and can be serviced easily when experiencing an error condition or failure.

- Reliability: It is an ability of the system to function correctly and ensure data integrity over a period without any failures to limit the probability of incorrect outcomes.
- Availability: It is an ability of the system to remain operational by either handling the event of errors or faults or by operating in the degraded mode by disabling the malfunctioned component or predicting the failure before occurring. This minimizes the application outages.
- Serviceability: It is the ease and pace with which a system can be maintained, repaired, or updated from a fault to be fully operational. This is done by having a detailed error logging to identify the occurrence of failure and performing repair with minimal disruption.

The reliability, availability and serviceability of the data-center server platforms requires to maximize the ease of system management for reliable operations, maintaining the customer service and reduce the cost of ownership of the platforms. The key aspects RAS in server systems-

- Error Detection, Correction and Containment
- Fault Isolation and Management
- RAS Offload to Baseboard Management Controller
- Platform Level Predictive Fault Analysis
- OS Level Predictive Fault Analysis
- Predictive Maintenance

The objective of the RAS is to correct as many faults as possible to optimize the system for an elevated level of resiliency. These RAS features apply to recovery mechanisms to reduce the impact of failure when an uncorrectable fault arises in the system. RAS is a complex architecture which amalgamates implementation of system with error reporting and logging mechanisms, error handling guidelines, error signaling flows.

3. Brief Understanding of Intel's Server Platforms

Data Center Platform	Supported CPU	Description
Eagle Stream	2 CPUs are supported. • Sapphire Rapids • Emerald Rapids	The legacy 4th Gen platform which aimed to provide the improvements in power management, security and manageability compared from previous generation.
Birch Stream	3 CPUs are supported. • Intel® Xeon® 6900/6700 series with PCore • Intel® Xeon® 6900/6700 series with ECore • Clearwater Forest	It is a high bandwidth datacenter platform which is available in two flavors (1) Advanced Performance core which supports 12 memory channels and up to 240 high-speed IO lanes. (2) Scalable Performance core which supports 8 memory channels and up to 184 high speed IO lanes.
Oak stream	• Diamond Rapids	It is a newer platform that includes updates and enhancements over previous platforms and improves platform monitoring and telemetry.

Note: Refer to Intel® Xeon® whitepaper releases for more details.

4. Processor Sub-Components and RAS Features

4.1 System RAS

System RAS refers to set of features and technologies designed to ensure that systems can operate reliably, are available, and are serviceable to be efficient such that data integrity is paramount.

4.1.1 System RAS Features

1) MCA Recovery – Execution Path

It is an OS assisted recovery mechanism used to recover from UC (uncorrectable) errors during software execution path. The main principle behind this is corrupt data containment which ensures that errors are contained and remain isolated to prevent further containment. At the time of execution, the micro-operation is prevented from retiring (containing the error). Errors are tagged with poison indicators that accompany the data. Recoverable errors are logged and reported with specific error signatures in MCA (Machine Check Architecture) banks and are signaled through MCE (Machine Check Exception). These UC errors as known as SRAR (Software Recoverable – Action Required) which require immediate action from the software to continue operating the system without any interruption. The recovery actions are taken by the software (OS/VMM, etc).

2) MCA Recovery – Non-Execution Path

It is an OS assisted recovery mechanism used to recover from UC errors that occur outside the direct execution path of software. MCA recovery in non-execution path enhances the reliability of the system by proving a flexible recovery option for errors. Although these SRAO (Software Recovery – Action Optional) occurs during program execution, they do not require immediate action and thus OS/VMM can defer the recovery actions if necessary. (Example: Errors include Patrol-Scrubbing errors, Last-Level Cache explicit write-back errors.) The software interprets the recovery action by analyzing the logged error signatures in corresponding registers associated with Machine Check Banks. Typically, the recovery action includes tagging affected memory pages as ‘bad’ ,handling them appropriately when accessed.

4.2 Memory RAS

Memory RAS encompasses a set of features and technologies designed to ensure that memory systems are reliable, available, and serviceable, minimizing downtime and data loss.

Fault Type	Possible Cause	Fault Coverage (RAS Features)
Bit/Cell Error	Soft Error can be due to high energy particle strike	SDDC, Demand Scrub, Patrol Scrub
Row Error	Persistent Faulty DRAM	ADDDC-SR, ADDDC-MR, SDDC, PPR
Bank Failure	Persistent Faulty DRAM	ADDDC-SR, ADDDC-MR, SDDC, PPR
Device Failure	Persistent Faulty DRAM	SDDC, ADDDC-MR, Rank Sparing
Address/Command Parity	Transient fault error on DDR link	DDR Cmd/Addr Parity error check and retry
Multi-Device Error	Persistent Faulty DRAM	MCA-recovery, Address Range Mirroring
DDR Channel Failure	Board Defect	Memory Map-out/Isolation

4.2.1 Memory RAS Features

1) <u>Memory Sparing</u>

Service costs of DIMMs are higher due to high rate of memory failures. Sparing is a technique to map out hard failures in the DRAM and thus reducing service costs. Sparing flows are:

a) post-package repair (PPR)

b) Adaptive Double Device Data Correction (ADDDC) - The flow is initiated based on the triggers setup by the software/BIOS/firmware based on the frequency of errors encountered in the channel.

1.(a) <u>Post Package Repair (PPR)</u>

It is an ability to map out the failing DRAM due to errors in bit/bank/column/device and using spare ECC device in its place. The usage model of PPR is:

i) <u>Boot-Time PPR</u> is a mechanism to repair faulty DRAM rows at boot time which are identified in the previous runtime (using Intel's proprietary methodology). Is makes use of the hard PPR features available in DDR's JEDEC specification. Boot time might be slightly extended, but it does not impact the performance of the system's operation.

ii) Runtime PPR is when DRAM's failing row is identified by the software and repaired at that time using sparing flow. This RAS feature is a primitive measure to prevent the system crash in place of waiting for repair on next boot cycle. It uses the available spare rows within the DRAM device to repair the fault while the system is still in operation.

1.(b) Adaptive Double Device Data Correction (ADDDC)

ADDDC is a memory RAS feature that allows runtime sparing by mapping out the failed DRAM device at bank/rank level granularity. ADDDC adapts the ECC scheme from SDDC to handle two device failures sequentially. The first failure is handled using device sparing and ECC scheme is adapted to handle the second failure. When the first device failure is detected, a sparing flow is initiated to map out the failed device and switch the ECC scheme from SDDC to ADDDC. This allows the system to correct the second device failure. ADDC uses Virtual Lockstep(VLS) to handle the failure. The cache line spreads across two ranks, where each device contains half and the new ECC. The lower half bursts of the cache line stay in the original rank and upper half bursts are swapped with the buddy rank and bank. This operation dynamically maps out faulty DRAM region and enhances the memory resiliency.

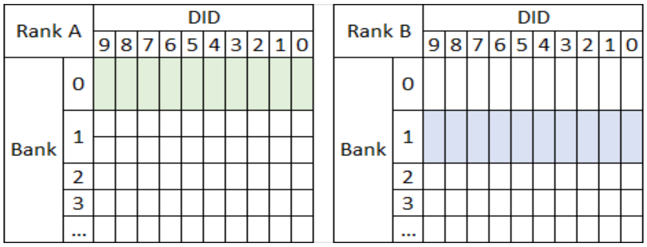
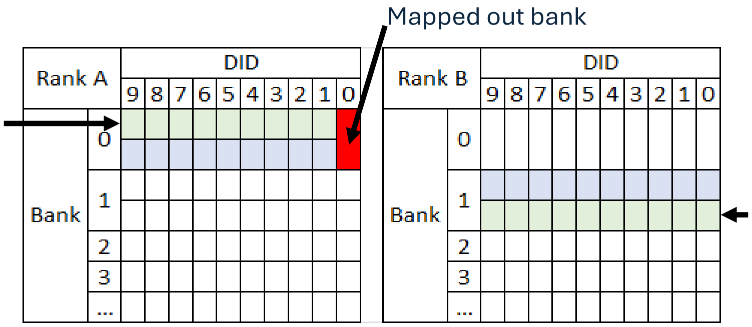


Figure: Data format before any VLS sparing



Lower half (bursts 0-7) of the line stays in the

Upper half (bursts 8-15) is swapped with the buddy rank and bank

Figure: Data format after bank VLS sparing

2) Memory Mirroring

It involves duplicating data across primary and secondary memory locations to ensure data integrity and makes the system reliable. In write operation - When the data is written to the memory, it is duplicated and written to both primary and secondary locations which ensures two copies of data is always available. During the read operation-reads are performed from primary memory location and if an uncorrectable error is detected, the system reads from the secondary memory location. When the data is read successfully from the secondary location, the systems perform memory scrub operation to test if error is soft error or due to hard failure. If the primary memory read returns the data without any error post scrub, the error is a soft error, else if the data is returned with an error, this is known as mirror failover and system issues an interrupt to notify this to BIOS/Firmware/OS.

3) Patrol Scrubbing

Patrol Scrubbing is the method to periodically read and verify the data stored in DRAM to prevent the accumulation of the correctable errors. The scrubbing operation involves the memory controller which periodically reads data from the memory, correcting any detected errors using ECC and writing back the corrected data to the same memory address. Steps in Patrol Scrubbing:

i) The patrol scrub engine reads data from the specified memory address.
ii) ECC is used to correct single-bit error when an error is detected.
iii) The corrected data is written to the memory address. If the detected error is an uncorrectable error, the data written is marked as poison.

The default scrubbing interval is 24-hour period but is programmable using BIOS knobs which writes the interval value to the dedicated register. Patrol Scrubbing can be performed after disabling the memory sparing operation to avoid the coherency issues.

4.3 PCIe RAS

PCIe (Peripheral Component Interconnect Express) is a packet-based transaction protocol standard for serial interface connecting high-speed general-purpose input-output interface (GPIOs). PCIe link provides a serial point–to–point communication channel between the end-point devices and CPU (via PCIe Topology).

4.3.1 PCIe Forward Error Correction (FEC)

PCIe 6 employs the PAM4 (Pulse Amplitude Modulation – 4 Level Voltage) encoding for the channel to have 64GT/s data rate per lane. This makes the link more susceptible to errors and can reduce the Bit Error Rate(BER) to 10^{-6} from standard allowance of 10^{-12}. To improve this and bring down the errors, PCIe 6 implements Forward Error Correction (FEC) using a Flow Control Unit (as known as FLIT). The Flit generates a payload of 256Byte whih includes a 6Byte of FEC, 8Byte of CRC and 243Byte of the remaining payload. In the Flit mode, Flex Bus Logical Block (FBLP) implements FCR block (FEC/CRC/Retry) at transmitter end (to pack payload with CRC and FEC code) and at receiver end (to check and compare the FEC and CRC of the payload to check for erroneous/faulty packets). The FCR block performs the flit packing and unpacking, FEC and CRC generation and checking, flit acknowledgement (ACKs and NAKs) generation and checking, and link level retry and retraining via retry buffers.

4.3.2 PCIe Error Types

1. Correctable Error: It includes the error conditions where a hardware can recover without any information loss. Hardware corrects these errors and intervention of BIOS/OS is not required (Ex: CRC errors in TLP).

2. Uncorrectable Error: These errors impact the functionality of the interface and reporting a UC error is analogous #SERR assertion.

2.1 Non-Fatal Error: This is a type of UC error which makes the specific transaction unreliable, but the link is fully functional. A reset of link might be needed to return to a reliable operation.

2.2 Fatal Error: This is a type of UC error which makes the hardware unreliable and relinquish with the link. A recovery from the error condition is necessary without resetting the component.

2.3 Advisory Non-Fatal Error: These uncorrectable errors do not require an immediate attention system reset or a recovery action. They provide an advisory message to the system/software/firmware by signaling an ERR_COR, indicating that error occurred is logged and should be addressed later without halting the ongoing event.

4.3.3 PCIe Error Flow and Reporting Mechanism (End-Point Flow)

Case 1: Reported Error is Correctable Error or an Advisory Non-Fatal Error

- Correctable Error (CE) Detected bit is set in the Device Status Register if CE error is encountered by the link.
- Unsupported Request (UR) Detected bit is set in the Device Status Register if the error is originated from an unsupported request by the end-point device.
- Corresponding bit is set in Correctable Error Status Register. (This is part of Advance Error Reporting Mechanism (AER)). If the error is masked, the corresponding bit in Correctable Error Mask Register would have been set and this will terminate the flow, else it will check for Advisory Non-Fatal Error (ANF Error).
- Bit field representing ANF error is set in Uncorrectable Error Status Register and if the error is unmasked in Uncorrectable Error Mask Register, the prefix and header reporting fields are updated. (This is part of Advance Error Reporting Mechanism (AER)).
- An "ERR_COR" message is signaled when the error is not a UR or UR reporting is not enabled in Device Control register, and Correctable error reporting is enabled in the Device Control Register.

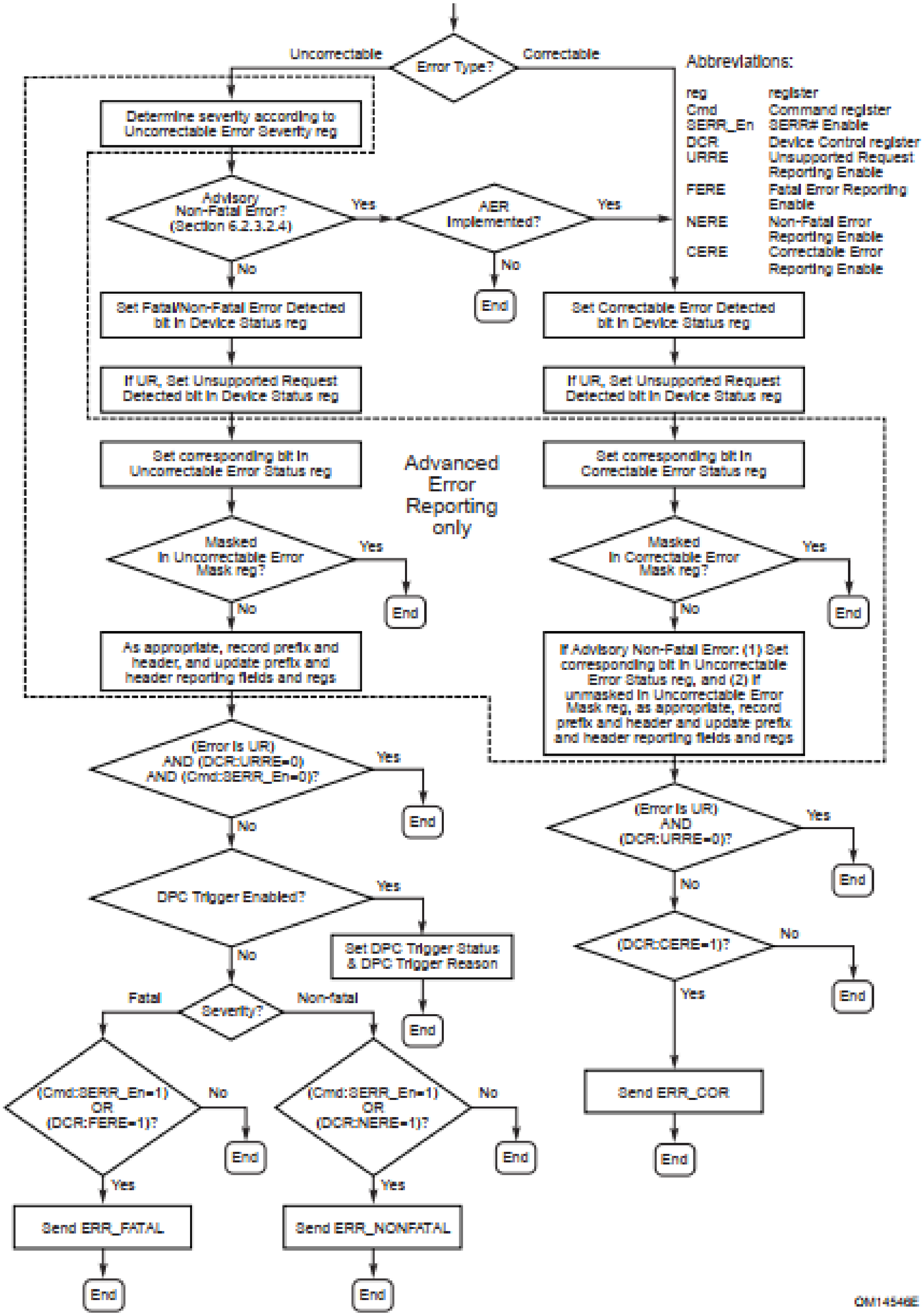
Case 2: Reported Error is Uncorrectable Error Non-Fatal Error

- Non-Fatal Detected bit is set in the Device Status Register if a non-fatal uncorrected error is encountered by the link.
- Unsupported Request (UR) Detected bit is set in the Device Status Register if the error is originated from an unsupported request by the end-point device.
- Corresponding bit is set in Uncorrectable Error Status Register. (This is part of Advance Error Reporting Mechanism (AER)). If the error is masked, the corresponding bit in Uncorrectable Error Mask Register would have been set and this will terminate the flow.
- Record the prefix and header reporting fields and update the register/reporting fields. (This is part of Advance Error Reporting Mechanism (AER)).
- An "ERR_NONFATAL" message is signaled when the following three conditions are met.

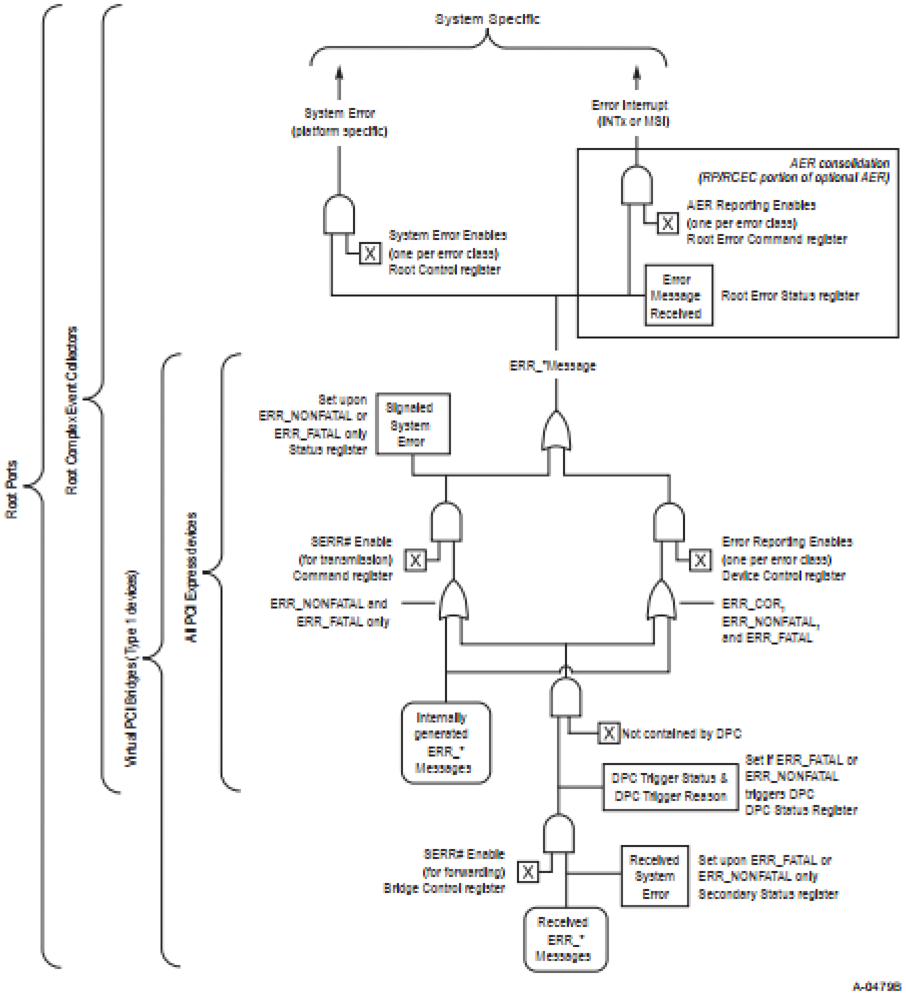
    - When this condition is not true (Error is a UR and UR reporting is disabled in Device Control register, and SERR_EN is disabled in Command Register).
    - DPC Triggering is disabled
    - SERR_EN is set in Command Register and Non-Fatal Error reporting is enabled.

Case 3: Reported Error is Uncorrectable Error Fatal Error

- Fatal Detected bit is set in the Device Status Register if a non-fatal uncorrected error is encountered by the link.
- Unsupported Request (UR) Detected bit is set in the Device Status Register if the error is originated from an unsupported request by the end-point device.
- Corresponding bit is set in Uncorrectable Error Status Register. (This is part of Advance Error Reporting Mechanism (AER)). If the error is masked, the corresponding bit in Uncorrectable Error Mask Register would have been set and this will terminate the flow.
- Record the prefix and header reporting fields and update the register/reporting fields. (This is part of Advance Error Reporting Mechanism (AER)).
- An "ERR_FATAL" message is signaled when the following three conditions are met.
    - When this condition is not true (Error is a UR and UR reporting is disabled in Device Control register, and SERR_EN is disabled in Command Register).
    - DPC Triggering is disabled
    - SERR_EN is set in Command Register and Fatal Error reporting is enabled.


Error Type?
Uncorrectable
Correctable
Abbreviations:
reg register
Cmd Command register
SERR_En SERR# Enable
DCR Device Control register
URRE Unsupported Request Reporting Enable
FERE Fatal Error Reporting Enable
NERE Non-Fatal Error Reporting Enable
CERE Correctable Error Reporting Enable
Determine severity according to Uncorrectable Error Severity reg
Advisory Non-Fatal Error? (Section 6.2.3.2.4)
Yes
AER Implemented?
Yes
No
End
No
Set Fatal/Non-Fatal Error Detected bit in Device Status reg
Set Correctable Error Detected bit in Device Status reg
If UR, Set Unsupported Request Detected bit in Device Status reg
If UR, Set Unsupported Request Detected bit in Device Status reg
Set corresponding bit in Uncorrectable Error Status reg
Advanced Error Reporting only
Set corresponding bit in Correctable Error Status reg
Masked in Uncorrectable Error Mask reg?
Yes
End
No
Masked in Correctable Error Mask reg?
Yes
End
No
As appropriate, record prefix and header, and update prefix and header reporting fields and regs
If Advisory Non-Fatal Error: (1) Set corresponding bit in Uncorrectable Error Status reg, and (2) if unmasked in Uncorrectable Error Mask reg, as appropriate, record prefix and header and update prefix and header reporting fields and regs
(Error is UR) AND (DCR:URRE=0) AND (Cmd:SERR_En=0)?
Yes
End
No
(Error is UR) AND (DCR:URRE=0)?
Yes
End
No
DPC Trigger Enabled?
Yes
Set DPC Trigger Status & DPC Trigger Reason
End
No
(DCR:CERE=1)?
No
End
Yes
Severity?
Fatal
Non-fatal
(Cmd:SERR_En=1) OR (DCR:FERE=1)?
No
End
Yes
Send ERR_FATAL
End
(Cmd:SERR_En=1) OR (DCR:NERE=1)?
No
End
Yes
Send ERR_NONFATAL
End
Send ERR_COR
End
OM14546E

4.3.4 PCIe Error Flow and Reporting Mechanism (Root Port Flow)


System Specific
System Error (platform specific)
Error Interrupt (INTx or MSI)
AER consolidation (RP/RCEC portion of optional AER)
AER Reporting Enables (one per error class) Root Error Command register
System Error Enables (one per error class) Root Control register
Error Message Received
Root Error Status register
ERR_*Message
Set upon ERR_NONFATAL or ERR_FATAL only Status register
Signaled System Error
SERR# Enable (for transmission) Command register
Error Reporting Enables (one per error class) Device Control register
ERR_NONFATAL and ERR_FATAL only
ERR_COR, ERR_NONFATAL, and ERR_FATAL
Internally generated ERR_* Messages
Not contained by DPC
DPC Trigger Status & DPC Trigger Reason
Set if ERR_FATAL or ERR_NONFATAL triggers DPC DPC Status Register
SERR# Enable (for forwarding) Bridge Control register
Received System Error
Set upon ERR_FATAL or ERR_NONFATAL only Secondary Status register
Received ERR_* Messages
A-0479B
Root Ports
Root Complex Event Collectors
Virtual PCI Bridges (Type 1 devices)
All PCI Express devices

4.3.5 PCIe RAS Features

1) <u>PCIe Link CRC and FEC Error Check and Recovery</u>

High speed interconnects are affected by many issues causing the signal integrity issues. In Flit mode, the receiver checks for the FEC error first in the received packet. If the FEC calculated at the receiver matches to the value calculated at the time of transmission, it checks for CRC on the packet. (In the non-Flit mode, only CRC is checked.)

2) <u>PCIe Link Retraining and Recovery</u>

A TLP packet sent over a link has a CRC calculated value by transmitter and a unique sequence number.

At the transmitter side

- The sequence number of the packet is obtained (assigned by the counter).
- A 32-bit LCRC is calculated based on the header and data to be transmitted, appends the generated LCRC value to the same packet.
- Whenever transmitter receives the ACK, it discards the packets from the retry buffer with sequence number less than / equal to the ACK received against the sequence number of the packet.

At the receiver (based on CRC and sequence number, when check pass)

- LCRC is calculated and checked with CRC value with packet. If the LCRC is matched, check passes with no transmission error.
- Post this LCRC check, it looks at the sequence number of the packet (the packets are transmitted in sequence). The sequence number of the packet is checked against the expected sequence number.

At the receiver (when check fails)

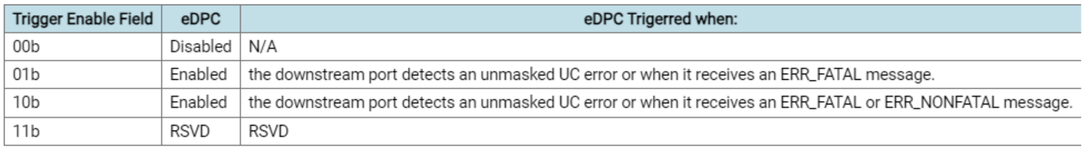
- When the LCRC or sequence number check fails, a NAK is generated with the last good sequence number of TLP received.
- This NAK is sent once the Ack/Nak latency timer expires which allows other TLPs to complete and thus a single ACK/NAK can be sent to enhance the efficiency.

3) ECRC

Switch is responsible to alter some control fields (if needed) without changing the other fields in the packet. When the TLP passes through switch, it regenerates the CRC code for the packet with updated control fields. However, the data corruption occurring internally in switch is masked out during the CRC regeneration. Thus, some systems require a highly reliable end-to-end data path protection. A 32-Bit ECRC in the TLP Digest fields at the end of TLP provides this support and ECRC covers all fields that do not change.(Only 2 fields can change their value, Bit 0 (change in this bit is due to forwarding of the configuration transaction across the bridge), and EP bit (change is when an UC error is encountered on data in the TLP progress path). ECRC is generated by the transmitters' Transaction Layer and verified by the PCIe receiver. (Optionally, ECRC can be checked by the intermediate receivers but must be untouched to the destination).

4) Enhance Downstream Port Containment (EDPC)

EDPC is an optional implementation feature and is disabled by default. It is enabled by the software when triggered. This RAS features allows to halt the PCIe traffic below the downstream port post an unmasked UC error detection at or below the port. This supports containment error recovery which is implemented by the software/BIOS/firmware.



Trigger Enable Field	eDPC	eDPC Trigerred when:
00b	Disabled	N/A
01b	Enabled	the downstream port detects an unmasked UC error or when it receives an ERR_FATAL message.
10b	Enabled	the downstream port detects an unmasked UC error or when it receives an ERR_FATAL or ERR_NONFATAL message.
11b	RSVD	RSVD

The Flow of EDPC

- When the eDPC is triggered, the downstream port sets DPC Trigger Status Bit (in the dedicated register) and the reason field which caused DPC triggering.(unmasked UC, non-fatal error, fatal error, root port programmed IO (RP PIO), SW triggered).
- This disables link by directing the LTSSM to disable state and stays until the DPC Triggered Status bit is cleared.
- If the DPC root port busy bit is set, then it waits for busy bit to be cleared to 0 and this is the time to quiesce and clean internal activities like DMA read request.
- Software releases the downstream port and LTSSM goes to detect state, link attempts to retrain.

The Link Retraining Mechanism

- Software can use one or both of the following to signal when the link reaches the DL_Active state again, Data Link Layer State Changed Interrupts, DL_Active Err_Cor Signaling.
- Support for DL_ACTIVE ERR_COR signaling is indicated via a bit DL_ACTIVE_ERR_COR in the DPC capability register.
- It is enabled by setting DL_ACTIVE ERR_COR_Enable bit in the DPC control register.
- The DL_ACTIVE state is indicated by the Data Link Layer Link Active bit in the Link Status register. DL_ACTIVE ERR_COR signaling is managed

independently of Data Link Layer State Changed interrupts, and it is permitted to use both mechanisms concurrently. If the DL_ACTIVE_ERR_COR_Enable bit is set and the Correctable Error Reporting Enable bit in the Device Control register is set, the port must send an ERR_COR message each time the link transitions into the DL_ACTIVE state. Since this event is not handled as an error, the Correctable Error Detected bit in the Device Status Register must not be set due to DL_ACTIVE_ERR_COR signaling. DL_ACTIVE_ERR_COR is signaled only when the link enters DL_ACTIVE state.

- DL_ACTIVE event, if a Port is going to send both an ERR_COR Message and an MSI/MSI-X transaction, then the Port must send the ERR_COR Message prior to sending the MSI/MSI-X transaction. There is no corresponding requirement if the INTx mechanism is being used to signal DL_ACTIVE interrupts, since INTx Messages won't necessarily remain ordered with respect to ERR_COR Messages when passing through routing elements. It is recommended that Operating Systems use the Data Link Layer State Changed Interrupts method for signaling when DL_ACTIVE changes state. The DL_ACTIVE_ERR_COR signaling method represents a subset of the same events and is primarily intended for use by platform SW when it needs to be notified to do Downstream Port configuration or provide Firmware First services.

RP PIO (Root Port Programmed IO) Error Handling

- All non-posted requests from Host Processor or peer-to-peer request from other root ports or req from RP Integrated endpoints are tracked and controlling operation by RP PIO error control register.
- UC or ANF errors encountered by RP PIO
    - Unsupported Req Status (UR Completion)
    - Completion with Completer Abort status (CA Completion)
    - Completion Timeout (CTO)
    - Control and status, per-request type specified as Configuration register, I/O request, memory request)
- Example: UR Completion error with Memory request triggers DPC for proper containment
- Handling of RP PIO error is determined by RP PIO Severity register
    - If Severity Bit = 1, error is handled as UC
    - if DPC Enable bit = 1

    - triggers DPC
    - DPC Interrupt
    - and/or ERR_COR is signaled

Note: Enhanced Root Port Error Reporting provides the capability to log UR, CA and CTO errors for Memory, IO and Config requests. RPPIO Header log captures the failing completion header. This is a vendor defined capability.

5. Machine Check Architecture and Signaling

The Machine Check Architecture (MCA) is Intel's robust framework that provides a comprehensive error detection, logging, and signaling architecture. It is an architectural framework that provides a mechanism for the hardware to communicate the occurrence of errors to BIOS/OS and ensures that appropriate action can be taken to maintain the system stability.

The MCA consists of multiple banks (each bank is associated with specific hardware unit or error source). Each of these banks contains a set of control and status Model-Specific Registers (MSRs) used for logging and managing the errors. These register set includes registers associated to function/features like, enabling/disabling the error reporting, logging error information, logging the address of error, type of errors, and additional error handling capabilities.

The BIOS/OS is notified of the detected errors by MCA signaling to take corrective actions. The signaling includes:

1) Machine Check Exception (MCE): The uncorrectable errors that are critical and require immediate action are signaled through MCE. This interrupt causes the CPU to transition to error handling routine.
2) System Management Interrupt (SMI): SMI signals the uncorrectable errors are processed by SMM(System Management Mode) BIOS which may involve error logging, error correction, and hiding the error from the OS. SMIs have highest priority and can prevent the normal operations from happening to manage critical errors.
3) Machine Check System Management Interrupt (MSMI): Enhanced MCA (eMCA) introduces MSMI for handling uncorrectable errors through SMM Bios. This allows firmware to process and mitigate errors before the OS is notified.
4) Corrected Machine Check Interrupt (CMCI): CMCI is an OS visible external interrupt and is used to signal Correctable errors (CE) or Uncorrectable Error – No action required (UCNA. This signaling do not require immediate attention but are logged for monitoring and threshold-based actions.
5) Correctable System Management Interrupt (CSMI): Correctable errors (CE) or Uncorrectable Error – No action required (UCNA) can also be signaled using CSMI, allows the SMM Bios to manage these errors.

6. Learnings from Testing and Debugging

6.1 Error Spoofing

Error Spoofing is an asynchronous machine-check error injection(AMEI) mechanism which is used to spoof errors without the need for actual hardware faults and allows software to write specific values into MCA registers. This is useful for testing and validating error handling flows in the software ensuring that the system behaves correctly under various fault conditions. The system generates an MCA message based on error spoofed and message is processes like real hardware error has occurred.

6.2 CATERR# (Catastrophic Error) Pin and Signaling

Caterr pin is used for signaling critical error conditions in Intel architecture. It is primarily used to notify the platform about severe error conditions that require immediate attention. The #CATERR pin covers three main error types based on signaling pulse on the pin.

1) MCERR (Machine Check Error): This error is a fatal error signaled when the CATERR# pin has 16 clock pulse. The condition occurs when the system detects a non-recoverable error that does not require an immediate shutdown. The pulse indicates the platform that a machine check error has occurred, and platform prompts appropriate error handling routines.
2) IERR: This is a fatal error signaled when the CATERR# pin is level signaled (asserted low) until the system resets. The error condition is triggered when the system detects a catastrophic unrecoverable error (example: Three-strike Timeout). The level signal held low indicates a critical error requiring an immediate system reset.

6.3 RMCA Error Pin and Signaling

Recoverable Machine Check Error are errors indicating software recoverable uncorrectable error (SRAR or SRAO) that are signaled when the RMCA# pin has 16 clock pulse. RMCA errors are notified by the RSMI/RMSMI interrupt, and the valid error signature is logged in MCA banks.

6.4 Behavior of Patrol Scrubbing in presence of continuous Uncorrectable Error Injection

When continuous error injected, the Patrol Scrubber keep reporting. Patrol scrub reads are partial reads. There will be a write associated with that read. The data will not change but to scrub it reads and writes. This triggers and SMI each time when the error is reported.

6.5 Response of System when a PMIC Error occurs in the system

When a system supports DDR5 PMIC (Power Management Integrated Circuit), BIOS will support the DIMM Failure Isolation. When the system encounters a PMIC power failure (e.g. critical temperature shutdown event), system responds in following two phases:

a. Phase 1 – The system discovers the DIMM where the event has occurred and isolates the bad PMIC. BIOS disables DIMM channel corresponding to the failure event.
b. Phase 2 – System Bios checks the status to ensure all PMICs expected to power-on the DIMM can do it successfully.

7. Conclusion

In conclusion, my internship experience at INTEL Corporation provided me with a deep understanding of key features related to RAS (Reliability, Availability, and Serviceability) in modern computing systems, with a specific focus on Systems, Memory, and PCIe links. I gained firsthand experience testing CPUs on the Birch Stream Platform, where I observed how errors are logged, signaled, and recovered through RAS actions. This insight into error detection and recovery mechanisms was invaluable in understanding how system failures are mitigated and how to ensure high system reliability.

A significant part of my internship involved collaborating with senior platform application engineers (PAEs) to replicate customer and internal issues. I actively assisted in troubleshooting and root cause analysis, contributing to reporting key findings and reasons behind complex system failures. This hands-on experience enhanced my critical thinking and problem-solving skills while deepening my understanding of how cross-functional teams work together to maintain system stability. Additionally, I helped create recipes for error spoofing and PMIC failure isolation, as well as testing existing recipes for integration and validation guides. These contributions strengthened my technical skills and allowed me to play a key role in ensuring the accuracy and reliability of test protocols.

Moreover, I contributed to preparing customer training materials for Xeon platforms, which not only helped me refine my technical writing abilities but also ensured that the knowledge I gained throughout the internship could be effectively communicated to those interacting with customers or using the platform in production environments.

Overall, this internship was a highly rewarding experience that allowed me to apply theoretical knowledge to real-world scenarios, develop critical technical skills, and work alongside experienced professionals in a dynamic, challenging environment. The exposure to RAS features and firsthand experience with error logging, recovery mechanisms, and platform testing has deepened my understanding of complex system architectures. This experience has also reinforced my career interest in systems engineering and platform development, and I look forward to applying the knowledge and skills gained during this internship in my future academic and professional endeavors.

8. References

For more details, refer to:

- Intel® Xeon® Whitepaper "https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2023-05/reliability-availability-and-serviceability-xeon-ras-tech-paper.pdf".
- PCIe Gen6 Base Specification for PCIe Subcomponent and PCIe RAS features – The specification can be downloaded from "https://pcisig.com/specifications" .