Enable PCI Express Advanced Error Reporting in the Kernel

Yanmin Zhang and T. Long Nguyen
Intel Corporation
yanmin.zhang@intel.com, tom.l.nguyen@intel.com

Abstract

PCI Express is a high-performance, general-purpose I/O Interconnect. It introduces AER (Advanced Error Reporting) concepts, which provide significantly higher reliability at a lower cost than the previous PCI and PCI-X standards. The AER driver of the Linux kernel provides a clean, generic, and architecture-independent solution. As long as a platform supports PCI Express, the AER driver shall gather and manage all occurred PCI Express errors and incorporate with PCI Express device drivers to perform error-recovery actions.

This paper is targeted toward kernel developers interested in the details of enabling PCI Express device drivers, and it provides insight into the scope of implementing the PCI Express AER driver and the AER conformation usage model.

1 Introduction

Current machines need higher reliability than before and need to recover from failure quickly. As one of failure causes, peripheral devices might run into errors, or go crazy completely. If one device is crazy, device driver might get bad information and cause a kernel panic: the system might crash unexpectedly.

As a matter of fact, IBM engineers (Linas Vepstas and others) created a framework to support PCI error recovery procedures in-kernel because IBM Power4 and Power5-based pSeries provide specific PCI device error recovery functions in platforms [4]. However, this model lacks the ability to support platform independence and is not easy for individual developers to get a Power machine for testing these functions. The PCI Express introduces the AER, which is a world standard. The PCI Express AER driver is developed to support the PCI Express AER. First, any platform which supports the PCI Express could use the PCI Express AER driver to process device errors and handle error recovery accordingly. Second, as lots of platforms support the PCI Express, it is far easier for individual developers to get such a machine and add error recovery code into specific device drivers.

2 PCI Express Advanced Error Reporting Driver

2.1 PCI Express Advanced Error Reporting Topology

To understand the PCI Express Advanced Error Reporting Driver architecture, it helps to begin with the basics of PCI Express Port topology. Figure 1 illustrates two types of PCI Express Port devices: the Root Port and the Switch Port. The Root Port originates a PCI Express Link from a PCI Express Root Complex. The Switch Port, which has its secondary bus representing switch internal routing logic, is called the Switch Upstream Port. The Switch Port which is bridging from switch internal routing buses to the bus representing the downstream PCI Express Link is called the Switch Downstream Port. Each PCI Express Port device can be implemented to support up to four distinct services: native hot plug (HP), power management event (PME), advanced error reporting (AER), virtual channels (VC).

The AER driver development is based on the service driver framework of the PCI Express Port Bus Driver design model [3]. As illustrated in Figure 2, the PCI Express AER driver serves as a Root Port AER service driver attached to the PCI Express Port Bus driver.

2.2 PCI Express Advanced Error Reporting Driver Architecture

PCI Express error signaling can occur on the PCI Express link itself or on behalf of transactions initiated on the link. PCI Express defines the AER capability, which is implemented with the PCI Express AER Extended

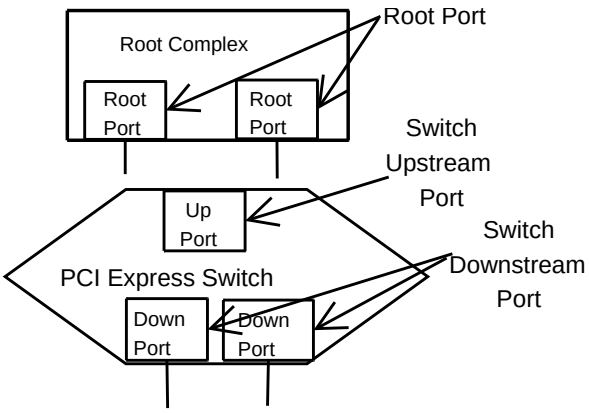

Figure 1: PCI Express Port Topology

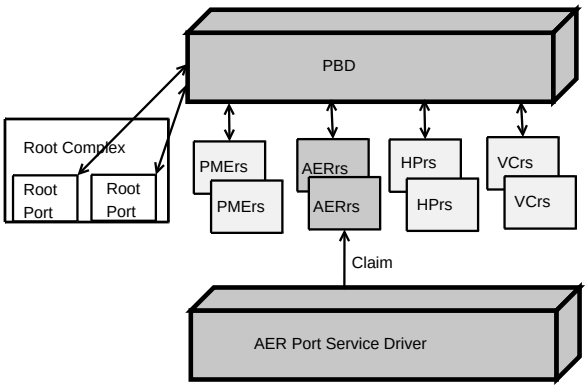

Figure 2: AER Root Port Service Driver

Capability Structure, to allow a PCI Express component (agent) to send an error reporting message to the Root Port. The Root Port, a host receiver of all error messages associated with its hierarchy, decodes an error message into an error type and an agent ID and then logs these into its PCI Express AER Extended Capability Structure. Depending on whether an error reporting message is enabled in the Root Error Command Register, the Root Port device generates an interrupt if an error is detected. The PCI Express AER service driver is implemented to service AER interrupts generated by the Root Ports. Figure 3 illustrates the error report procedures.

Once the PCI Express AER service driver is loaded, it claims all AERrs service devices in a system device hierarchy, as shown in Figure 2. For each AERrs service device, the advanced error reporting service driver configures its service device to generate an interrupt when an error is detected [3].

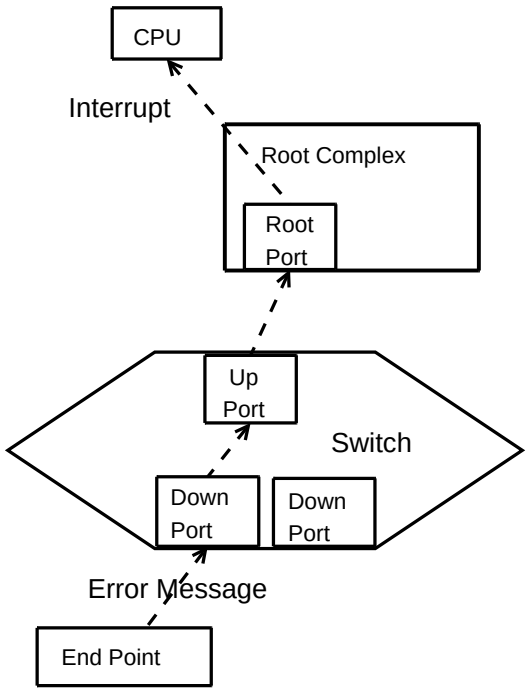

Figure 3: PCI Express Error Reporting procedures

When errors happen, the PCI Express AER driver could provide such infrastructure with three basic functions:

- Gathers the comprehensive error information if errors occurred.
- Performs error recovery actions.
- Reports error to the users.

2.2.1 PCI Express Error Introduction

Traditional PCI devices provide simple error reporting approaches, PERR# and SERR#. PERR# is parity error, while SERR# is system error. All non-PERR# errors are SERR#. PCI uses two independent signal lines to represent PERR# and SERR#, which are platform chipset-specific. As for how software is notified about the errors, it totally depends on the specific platforms.

To support traditional error handling, PCI Express provides baseline error reporting, which defines the basic error reporting mechanism. All PCI Express devices have to implement this baseline capability and must map required PCI Express error support to the PCI-related error registers, which include enabling error reporting

and setting status bits that can be read by PCI-compliant software. But the baseline error reporting doesn't define how platforms notify system software about the errors.

PCI Express errors consist of two types, correctable errors and uncorrectable errors. Correctable errors include those error conditions where the PCI Express protocol can recover without any loss of information. A correctable error, if one occurs, can be corrected by the hardware without requiring any software intervention. Although the hardware has an ability to correct and reduce the correctable errors, correctable errors may have impacts on system performance.

Uncorrectable errors are those error conditions that impact functionality of the interface. To provide more robust error handling to system software, PCI Express further classifies uncorrectable errors as fatal and non-fatal. Fatal errors might cause corresponding PCI Express links and hardware to become unreliable. System software needs to reset the links and corresponding devices in a hierarchy where a fatal error occurred. Non-fatal errors wouldn't cause PCI Express link to become unreliable, but might cause transaction failure. System software needs to coordinate with a device agent, which generates a non-fatal error, to retry any failed transactions.

PCI Express AER provides more reliable error reporting infrastructure. Besides the baseline error reporting, PCI Express AER defines more fine-grained error types and provides log capability. Devices have a header log register to capture the header for the TLP corresponding to a detected error.

Correctable errors consist of receiver errors, bad TLP, bad DLLP, REPLAY_NUM rollover, and replay timer time-out. When a correctable error occurs, the corresponding bit within the advanced correctable error status register is set. These bits are automatically set by hardware and are cleared by software when writing a "1" to the bit position. In addition, through the Advanced Correctable Error Mask Register (which has the similar bitmap like advanced correctable error status register), a specific correctable error could be masked and not be reported to root port. Although the errors are not reported with the mask configuration, the corresponding bit in advanced correctable error status register will still be set.

Uncorrectable errors consist of Training Errors, Data Link Protocol Errors, Poisoned TLP Errors, Flow Control Protocol Errors, Completion Time-out Errors, Completer Abort Errors, Unexpected Completion Errors, Receiver Overflow Errors, Malformed TLPs, ECRC Errors, and Unsupported Request Errors. When an uncorrectable error occurs, the corresponding bit within the Advanced Uncorrectable Error Status register is set automatically by hardware and is cleared by software when writing a "1" to the bit position. Advanced error handling permits software to select the severity of each error within the Advanced Uncorrectable Error Severity register. This gives software the opportunity to treat errors as fatal or non-fatal, according to the severity associated with a given application. Software could use the Advanced Uncorrectable Mask register to mask specific errors.

2.2.2 PCI Express AER Driver Designed To Handle PCI Express Errors

Before kernel 2.6.18, the Linux kernel had no root port AER service driver. Usually, the BIOS provides basic error mechanism, but it couldn't coordinate corresponding devices to get more detailed error information and perform recovery actions. As a result, the AER driver has been developed to support PCI Express AER enabling for the Linux kernel.

2.2.2.1 AER Initialization Procedures

When a machine is booting, the system allocates interrupt vector(s) for every PCI Express root port. To service the PCI Express AER interrupt at a PCI Express root port, the PCI Express AER driver registers its interrupt service handler with Linux kernel. Once a PCI Express root port receives an error reported from the downstream device, that PCI Express root port sends an interrupt to the CPU, from which the Linux kernel will call the PCI Express AER interrupt service handler.

Most of AER processing work should be done under a process context. The PCI Express AER driver creates one worker per PCI Express AER root port virtual device. Depending on where an AER interrupt occurs in a system hierarchy, the corresponding worker will be scheduled.

Most BIOS vendors provide a non-standard error processing mechanism. To avoid conflict with BIOS while handling PCI Express errors, the PCI Express AER

driver must request the BIOS for ownership of the PCI Express AER via the ACPI _OSC method, as specified in PCI Express Specification and ACPI Specification. If the BIOS doesn't support the ACPI _OSC method, or the ACPI _OSC method returns errors, the PCI Express AER driver's probe function will fail (refer to Section 3 for a workaround if the BIOS vendor does not support the ACPI _OSC method).

Once the PCI Express AER driver takes over, the BIOS must stop its activities on PCI Express error processing. The Express AER driver then configures PCI Express AER capability registers of the PCI Express root port and specific devices to support PCI Express native AER.

2.2.2.2 Handle PCI Express Correctable Errors

Because a correctable error can be corrected by the hardware without requiring any software intervention, if one occurs, the PCI Express AER driver first decodes an error message received at PCI Express root port into an error type and an agent ID. Second, the PCI Express AER driver uses decoded error information to read the PCI Express AER capability of the agent device to obtain more details about an error. Third, the PCI Express AER driver clears the corresponding bit in the correctable error status register of both PCI Express root port and the agent device. Figure 4 illustrates the procedure to process correctable errors. Last but not least, the details about an error will be formatted and output to the system console as shown below:

```
+—— PCI-Express Device Error ——+
Error Severity : Corrected
PCIE Bus Error type : Physical Layer
Receiver Error : Multiple
Receiver ID : 0020
VendorID=8086h, DeviceID=3597h, Bus=00h, Device=04h,
Function=00h
```

The Requester ID is the ID of the device which reports the error. Based on such information, an administrator could find the bad device easily.

2.2.2.3 Handle PCI Express Non-Fatal Errors

If an agent device reports non-fatal errors, the PCI Express AER driver uses the same mechanism as described in Section 2.2.2 to obtain more details about an error from an agent device and output error information to the system console. Figure 5 illustrates the procedure to process non-fatal errors.

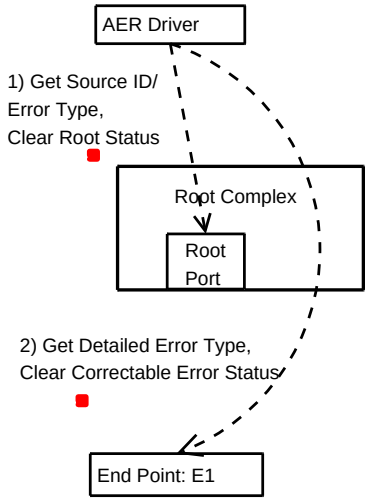

Figure 4: Procedure to Process Correctable Errors

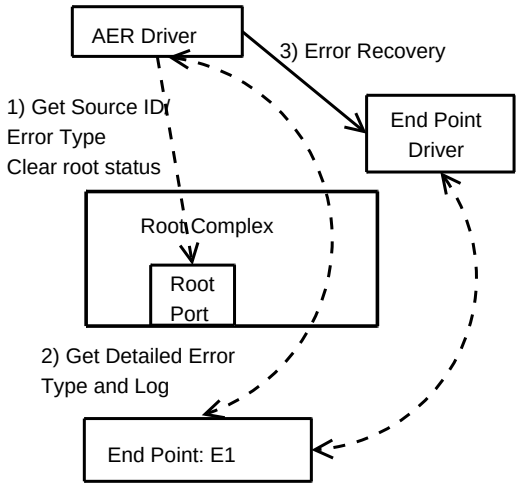

Figure 5: Procedures to Process Non-Fatal Errors

The first two steps are like the ones to process correctable errors. During Step 2, the AER driver need to retrieve the packet header log from the agent if the error is TLP-related.

Below is an example of non-fatal error output to the system console.

```
+—— PCI-Express Device Error ——+
Error Severity : Uncorrected (Non-Fatal)
PCIE Bus Error type : Transaction Layer
Completion Timeout : Multiple
Requester ID : 0018
VendorID=8086h, DeviceID=3596h, Bus=00h, Device=03h,
Function=00h
```

Unlike correctable errors, non-fatal errors might cause

some transaction failures. To help an agent device driver to retry any failed transactions, the PCI Express AER driver must perform a non-fatal error recovery procedure, which depends on where a non-fatal error occurs in a system hierarchy. As illustrated in Figure 6, for example, there are two PCI Express switches. If endpoint device E2 reports a non-fatal error, the PCI Express AER driver will try to perform an error recovery procedure only on this device. Other devices won't take part in this error recovery procedure. If downstream port P1 of switch 1 reports a non-fatal error, the PCI Express AER driver will do error recovery procedure on all devices under port P1, including all ports of switch 2, end point E1, and E2.

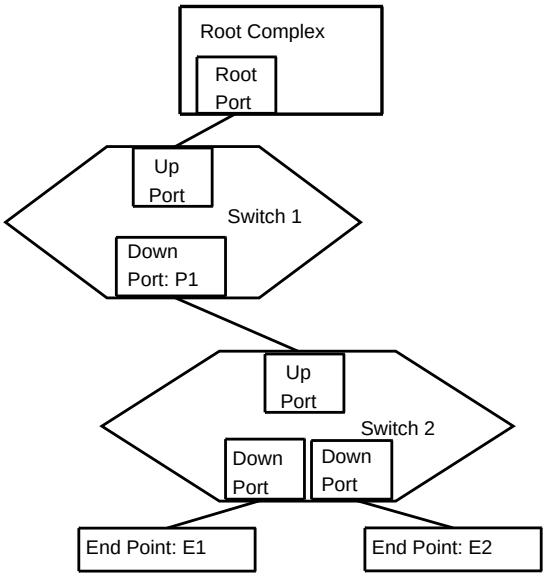

Figure 6: Non-Fatal Error Recovery Example

To take part in the error recovery procedure, specific device drivers need to implement error callbacks as described in Section 4.1.

When an uncorrectable non-fatal error happens, the AER error recovery procedure first calls the `error_detected` routine of all relevant drivers to notify their devices run into errors by the deep-first sequence. In the callback `error_detected`, the driver shouldn't operate the devices, i.e., do not perform any I/O on the devices. Mostly, `error_detected` might cancel all pending requests or put the requests into a queue.

If the return values from all relevant `error_detected` routines are `PCI_ERS_RESULT_CAN_RECOVER`, the AER recovery procedure calls all resume callbacks of the relevant drivers. In the resume functions, drivers could resume operations to the devices.

If an `error_detected` callback returns `PCI_ERS_RESULT_NEED_RESET`, the recovery procedure will call all `slot_reset` callbacks of relevant drivers. If all `slot_reset` functions return `PCI_ERS_RESULT_CAN_RECOVER`, the resume callback will be called to finish the recovery. Currently, some device drivers provide `err_handler` callbacks. For example, Intel's E100 and E1000 network card driver and IBM's POWER RAID driver.

The PCI Express AER driver outputs some information about non-fatal error recovery steps and results. Below is an example.

```
+—— PCI-Express Device Error ——+
Error Severity : Uncorrected (Non-Fatal)
PCIE Bus Error type : Transaction Layer
Unsupported Request : First
Requester ID : 0500
VendorID=14e4h, DeviceID=1659h, Bus=05h, Device=00h,
Function=00h
TLB Header:
04000001 0020060f 05010008 00000000
Broadcast error_detected message
Broadcast slot_reset message
Broadcast resume message
tg3: eth3: Link is down.
AER driver successfully recovered
```

2.2.2.4 Handle PCI Express Fatal Errors

When processing fatal errors, the PCI Express AER driver also collects detailed error information from the reporter in the same manner as described in Sections 2.2.2.2 and 2.2.2.3. Below is an example of non-fatal error output to the system console:

```
+—— PCI-Express Device Error ——+
Error Severity : Uncorrected (Fatal)
PCIE Bus Error type : Transaction Layer
Unsupported Request : First
Requester ID : 0200
VendorID=8086h, DeviceID=0329h, Bus=02h, Device=00h,
Function=00h
TLB Header:
04000001 00180003 02040000 00020400
```

When performing the error recovery procedure, the major difference between non-fatal and fatal is whether

the PCI Express link will be reset. If the return values from all relevant `error_detected` routines are `PCI_ERS_RESULT_CAN_RECOVER`, the AER recovery procedure resets the PCI Express link based on whether the agent is a bridge. Figure 7 illustrates an example.

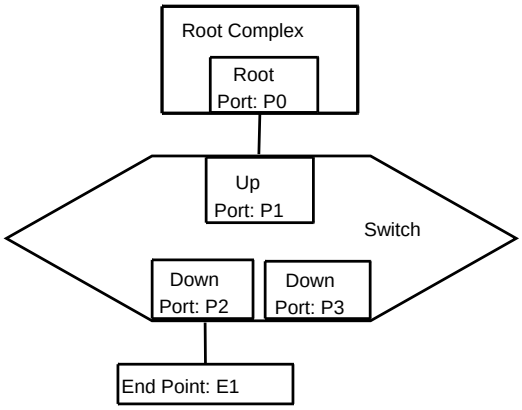

Figure 7: Reset PCI Express Link Example

In Figure 7, if root port P0 (a kind of bridge) reports a fatal error to itself, the PCI Express AER driver chooses to reset the upstream link between root port P0 and upstream port P1. If end-point device E1 reports a fatal error, the PCI Express AER driver chooses to reset the upstream link of E1, i.e., the link between P2 and E1.

The reset is executed by the port. If the agent is a port, the port will execute reset. If the agent is an end-point device, for example, E1 in Figure 7, the port of the upstream link of E1, i.e., port P2 will execute reset.

The reset method depends on the port type. As for root port and downstream port, the PCI Express Specification defines an approach to reset their downstream link. In Figure 7, if port P0, P2, P3, and end point E1 report fatal errors, the method defined in PCI Express Specification will be used. The PCI Express AER driver implements the standard method as default reset function.

There is no standard way to reset the downstream link under the upstream port because different switches might implement different reset approaches. To facilitate the link reset approach, the PCI Express AER driver adds `reset_link`, a new function pointer, in the data structure `pcie_port_service_driver`.

```
struct pcie_port_service_driver {
    ...
    /* Link Reset Capability - AER service
       driver specific */
    pci_ers_result_t (*reset_link) (struct
       pci_dev *dev);
    ...
};
```

If a port uses a vendor-specific approach to reset link, its AER port service driver has to provide a `reset_link` function. If a root port driver or downstream port service driver doesn't provide a `reset_link` function, the default `reset_link` function will be called. If an upstream port service driver doesn't implement a `reset_link` function, the error recovery will fail.

Below is the system console output example printed by the PCI Express AER driver when doing fatal error recovery.

```
+—— PCI-Express Device Error ——+
Error Severity : Uncorrected (Fatal)
PCIE Bus Error type : (Unaccessible)
Unaccessible Received : First
Unregistered Agent ID : 0500
Broadcast error_detected message
Complete link reset at Root[0000:00:04.0]
Broadcast slot_reset message
Broadcast resume message
tg3: eth3: Link is down.
AER driver successfully recovered
```

2.3 Including PCI Express Advanced Error Reporting Driver Into the Kernel

The PCI Express AER Root driver is a Root Port service driver attached to the PCI Express Port Bus driver. Its service must be registered with the PCI Express Port Bus driver and users are required to include the PCI Express Port Bus driver in the kernel [5]. Once the kernel configuration option `CONFIG_PCIEPORTBUS` is included, the PCI Express AER Root driver is automatically included as a kernel driver by default (`CONFIG_PCIEAER = Y`).

3 Impact to PCI Express BIOS Vendor

Currently, most BIOSes don't follow PCI FW 3.0 to support the ACPI _OSC handler. As a result, the PCI Express AER driver will fail when calling the ACPI

control method _OSC. The PCI Express AER driver provides a current workaround for the lack of ACPI BIOS _OSC support by implementing a boot parameter, `forceload=y/n`. When the kernel boots with parameter `aerdriver.forceload=y`, the PCI Express AER driver still binds to all root ports, which implements the AER capability.

4 Impact to PCI Express Device Driver

4.1 Device driver requirements

To conform to AER driver infrastructure, PCI Express device drivers need support AER capability.

First, when a driver initiates a device, it needs to enable the device's error reporting capability. By default, device error reporting is turned off, so the device won't send error messages to root port when it captures an error.

Secondly, to take part in the error recovery procedure, a device driver needs to implement error callbacks as described in the `pci_error_handlers` data structure as shown below.

```
struct pci_error_handlers {
  /* PCI bus error detected on this device */
  pci_ers_result_t (*error_detected)(struct
    pci_dev *dev, enum pci_channel_state error);
  /* MMIO has been re-enabled, but not DMA */
  pci_ers_result_t (*mmio_enabled)(struct
    pci_dev *dev);
  /* PCI slot has been reset */
  pci_ers_result_t (*slot_reset)(struct
    pci_dev *dev);
  /* Device driver may resume
    normal operations */
  void (*resume)(struct pci_dev *dev);
};
```

In data structure `pci_driver`, add `err_handler` as a new pointer to point to the `pci_error_handlers`. In kernel 2.6.14, the definition of `pci_error_handlers` had already been added to support PCI device error recovery [4]. To be compatible with PCI device error recovery, PCI Express device error recovery also uses the same definition and follows a similar rule. One of our starting points is that we try to keep the recovery callback interfaces as simple as we can. If the interfaces are complicated, there will be no driver developers who will be happy to add error recovery callbacks into device drivers.

4.2 Device driver helper functions

To communicate with device AER capabilities, drivers need to access AER registers in configuration space. It's easy to write incorrect code because they must access/change the bits of registers. To facilitate driver programming and reduce coding errors, the AER driver provides a couple of helper functions which could be used by device drivers.

4.2.1 int pci_find_aer_capability (struct pci_dev *dev);

`pci_find_aer_capability` locates the PCI Express AER capability in the device configuration space. Since offset 0x100 in configuration space, PCI Express devices could provide a couple of optional capabilities and they link each other in a chain. AER is one of them. To locate AER registers, software needs to go through the chain. This function returns the AER offset in the device configuration space.

4.2.2 int pci_enable_pcie_error_reporting (struct pci_dev *dev);

`pci_enable_pcie_error_reporting` enables the device to send error messages to the root port when an error is detected. If the device doesn't support PCI-Express capability, the function returns 0. When a device driver initiates a device (mostly, in its probe function), it should call `pci_enable_pcie_error_reporting`.

4.2.3 int pci_disable_pcie_error_reporting (struct pci_dev *dev);

`pci_disable_pcie_error_reporting` disables the device from sending error messages to the root port. Sometimes, device drivers want to process errors by themselves instead of using the AER driver. It's not encouraged, but we provide this capability.

4.2.4 int pci_cleanup_aer_uncorrect_error_status (struct pci_dev *dev);

`pci_cleanup_aer_uncorrect_error_status` cleans up the uncorrectable error status

register. The AER driver only clears correctable error status register when processing errors. As for uncorrectable errors, specific device drivers should do so because they might do more specific processing. Usually, a driver should call this function in its `slot_reset` or `resume` callbacks.

4.3 Testing PCI Express AER On Device Driver

It's hard to test device driver AER capabilities. By lots of experiments, we have found that UR (Unsupported Request) can be used to test device drivers. We triggered UR error messages by probing a non-existent device function. For example, if a PCI Express device only has one function, when kernel reads the ClassID from the configuration space of the second function of the device, the device might send an Unsupported Request error message to the root port and set the bit in uncorrectable error status register. By setting different values in the corresponding bit in uncorrectable error mask register, we could test both non-fatal and fatal errors.

5 Conclusion

The PCI Express AER driver creates a generic infrastructure to support PCI Express AER. This infrastructure provides the Linux kernel with an ability to capture PCI Express device errors and perform error recovery where in a hierarchy an agent device reports. Last but not least the system administrators could get formatted, useful error information to debug device errors.

Linux kernel 2.6.19 has accepted the PCI Express AER patches. Future work includes enabling PCI Express AER for every PCI Express device by default, blocking I/O when an error happens, and so on.

6 Acknowledgement

Special thanks to Steven Carbonari for his contributions to the architecture design of PCI Express AER driver, Rajesh Shah for his contributions to code review, and the Linux community for providing great input.

Legal Statement

References

[1] PCI Express Base Specification Revision 1.1. March 28, 2005. `http://www.pcisig.com`

[2] PCI Firmware Specification Revision 3.0, `http://www.pcisig.com`

[3] Tom Long Nguyen, Dely L. Sy, & Steven Carbonari. "PCI Express Port Bus Driver Support for Linux." *Proceedings of the Linux Symposium*, Vol. 2, Ottawa, Ontario, 2005. `http://www.linuxsymposium.org/2005/linuxsymposium_procv2.pdf`

[4] pci-error-recovery.txt. Available from: 2.6.20/Documentation.

[5] PCIEBUS-HOWTO.txt. Available from: 2.6.20/Documentation.

[6] pcieaer-howto.txt. Available from: 2.6.20/Documentation.

Proceedings of the Linux Symposium

Volume Two

June 27th–30th, 2007
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*