



## PCI-SIG ENGINEERING CHANGE NOTICE

<b>TITLE:</b>	Error Reporting
<b>DATE:</b>	January 27, 2005
<b>AFFECTED DOCUMENT:</b>	PCI Express Base Specification 1.0a
<b>SPONSOR:</b>	Joe Cowan; Hewlett-Packard Company

### **Part I**

#### **1. Summary of the Functional Changes**

PCIe is currently incompatible with PCI and PCI-X in the area of error reporting, in that PCIe requires a Completer that detects an uncorrectable error with a Non-Posted Request (NPR) to signal it via an uncorrectable error Message (if enabled) in addition to returning a UR/CA Status in the Completion. In some cases this precludes an NPR Requester from being able to determine how a resulting error condition will be handled. As a specific concern, if Unsupported Request (UR) Reporting is disabled to permit NPR probing during device enumeration, a Completer is unable to signal UR errors detected with Posted Requests (PRs).

This ECN restores PCIe compatibility with PCI and PCI-X in this respect, by having the Completer never signal NPR non-fatal errors with ERR\_NONFATAL. However, software can configure the Completer: (A) not to signal the error at all; (B) to signal the error with ERR\_COR as an advisory; or (C) to signal the error with ERR\_FATAL and bring down the hierarchy.

PCIe is also currently unable to meet its stated flexibility goals for handling TLPs with poison or bad ECRC, for reasons similar to those behind the problems with non-fatal NPRs. For example, a Switch that detects and signals a poisoned Memory Write Request using ERR\_NONFATAL may result in the PCIe hierarchy being brought down, even if the Completer of this PR doesn't want this to happen. This ECN addresses this concern with the same basic approach as above. An intermediate Receiver (e.g., a Switch) that detects poison or bad ECRC with a TLP will never signal it with ERR\_NONFATAL, but software can configure the intermediate Receiver with the same signaling choices as with non-fatal NPR errors (enumerated above).

The PCIe 1.0a spec intends for the PCIe-specific UR Reporting Enable bit, when clear, to prevent a Completer from ever signaling a UR error via an error Message, even when legacy PCI control bits enable SERR#. This makes it impossible for a Completer to signal PR UR errors with an error Message when configured by legacy PCI software, another incompatibility with PCI and PCI-X. This ECN restores that ability.

Finally, this ECN clarifies error reporting requirements for downstream ports when their associated link is down.

#### **2. Benefits as a Result of the Changes**

1. Software can configure a system where the Requester for an NPR with a Completer-detected non-fatal error is able to determine how the error condition will be handled, restoring compatibility in this area with PCI and PCI-X.
2. TLP poison or bad ECRC detection/reporting by intermediate Receivers no longer need interfere with more appropriate error handling by the ultimate Receiver.

3. UR reporting does not need to be disabled in order to support NPR probing during device enumeration. This permits URs to be reported for PRs, closing a potential source for silent data corruption, and also restoring compatibility in this area with PCI and PCI-X.
4. Legacy PCI software is able to enable UR error signaling for PRs, restoring compatibility with PCI and PCI-X.

### 3. Assessment of the Impact

1. Completer logic must change slightly in its handling of non-fatal errors with NPRs.
2. Intermediate Receiver (e.g., Switch) logic that detects non-fatal errors with TLPs (like poison or bad ECRC) must change slightly in its signaling of them.
3. The SERR# Enable bit in the Command register now implicitly enables UR reporting for uncorrectable error cases, even when the UR Reporting Enable bit in the Device Control register is clear.
4. Control logic in Virtual PCI Bridges (RPs & Switches) must change slightly regarding the transmission of error Messages that have been forwarded from the secondary interface to the primary interface.
5. For devices implementing AER, there is a new *Advisory Non-Fatal Error* bit in the Correctable Error Status and Mask registers.
6. A new capability bit, *Role-Based Error Reporting*, has been assigned in the Device Capability register to indicate to software if a device implements this ECN.

### 4. Analysis of the Hardware Implications

No impact to hardware interoperability.

### 5. Analysis of the Software Implications

Functional compatibility with legacy PCI software is increased. With components implementing this ECN, systems running software that enables SERR# automatically gain UR reporting for PRs.

Non-fatal errors with NPRs are no longer signaled with ERR\_NONFATAL. PCIe-aware software that leaves UR Reporting disabled to permit probing is not broken, but must enable UR Reporting if it wishes to take advantage of UR reporting with PRs. Such software probably should not enable UR reporting for components that do not implement this ECN's functionality. Software can determine if a component implements this ECN's functionality by examining a new capability bit.

Poison or bad ECRC TLP errors detected by intermediate Receivers are no longer signaled with ERR\_NONFATAL. This should not break legacy PCI software and is unlikely to break existing PCIe-aware software. New PCIe-aware software can enable Advisory Non-Fatal Error signaling (ERR\_COR) by intermediate Receivers if desired.

New components with AER have the *Advisory Non-Fatal Error* bit masked by default, to provide compatibility with 1.0a PCIe-aware software.

## Part II

### Detailed Description of the change

*Note: Specification text is based on Rev 1.1RD, published July 16,2004, which incorporates numerous 1.0a ECNs and Errata. All page numbers are based on that spec. Each section that includes text modified from Base Spec 1.0a is noted.*

*Change Terms and Acronyms Section, starting page 21 as follows:  
(Note: this section already incorporates 1.0a ECN/Errata text.)*

## Terms and Acronyms

...

Completer Abort, CA	1. A status that applies to a posted or non-posted Request that the Completer is permanently unable to complete successfully, due to a violation of the Completer's programming model or to <del>a fatal-an</del> <a href="#">unrecoverable</a> error associated with the Completer. 2. A status indication returned with a Completion for a non-posted Request that suffered a Completer Abort at the Completer.
<a href="#">error detection</a>	<a href="#">Mechanisms that determine that an error exists, either by the first agent to discover the error (e.g. Malformed TLP) or by the recipient of a signaled error (e.g. receiver of a poisoned TLP)</a>
<a href="#">error logging</a>	<a href="#">A detector setting one or more bits in architected registers based on the detection of an error. The detector might be the original discoverer of an error or a recipient of a signaled error.</a>
<del>Error Recovery, Error Detection</del>	<del>The mechanisms that ensure integrity of data transfer, including the management of the transmit side retry buffer(s).</del>
<a href="#">error reporting</a>	<a href="#">In a broad context, the general notification of errors. In the context of the Device Control register, sending an Error Message. In the context of the Root Error Command register, signaling an interrupt as a result of receiving an Error Message.</a>
<a href="#">error signaling</a>	<a href="#">One agent notifying another agent of an error either by (1) sending an Error Message, (2) sending a Completion with UR/CA Status, or (3) poisoning a TLP</a>
<a href="#">Reported Error</a>	<a href="#">An error subject to the logging and signaling requirements architecturally defined in this document</a>

*Change Section 2.3.2, page 99 as follows:*

### 2.3.2 Completion Handling Rules

...

- Completions with a Completion Status other than Successful Completion, or Configuration Request Retry Status (in response to Configuration Request only) must cause the Requester to:
  - Free Completion buffer space and other resources associated with the Request.
  - [Handle the error via a Requester-specific mechanism \(See Section 6.2.3.2.5\).](#)

Change Section 2.7.1, page 123 as follows:

## 2.7.1 ECRC Rules

...

Note that a Switch may perform ECRC checking on TLPs passing through the Switch. ECRC Errors detected by the Switch are reported ~~in the same way any other device would report them~~ as described in Table 6-7, but do not alter the TLPs passage through the Switch.

Change Section 2.9.1, page 130 as follows:

## 2.9.1 Transaction Layer Behavior in DL\_Down Status

...

For a Root Complex, or any Port on a Switch other than the one closest to the Root Complex, DL\_Down status is handled by:

...

- ❑ for Non-Posted Requests, forming completions for any Requests submitted by the device core for Transmission, returning Unsupported Request Completion Status, then discarding the Requests
  - This is a reported error associated with the device/function for the (virtual) Bridge associated with the Port (see Section 6.2). For Root Ports, the reporting of this error is optional.
  - Non-Posted Requests already being processed by the Transaction Layer, for which it may not be practical to return Completions, are discarded

Note: This is equivalent to the case where the Request had been Transmitted but not yet Completed before the Link status became DL\_Down

- ◆ These cases are handled by the Requester using the Completion Timeout mechanism

Note: The point at which a Non-Posted Request becomes “uncompletable” is implementation specific.

- ❑ for Posted Requests, discarding the Requests

- This is a reported error associated with the device/function for the (virtual) Bridge associated with the Port (see Section 6.2), and must be reported as an Unsupported Request. For Root Ports, the reporting of this error is optional.
- For a Posted Request already being processed by the Transaction Layer, the Port is permitted not to report it.

Note: This is equivalent to the case where the Request had been Transmitted before the Link status became DL\_Down

Note: The point at which a Posted Request becomes “unreportable” is implementation specific.

Change beginning at Section 6.2.2.1, page 291 as follows:

### 6.2.2.1 Correctable Errors

Correctable errors include those error conditions where the PCI Express protocol can recover without any loss of information. Hardware corrects these errors and software intervention is not required. For example, an LCRC error in a TLP ~~which is that might be~~ corrected by Data Link Level Retry is considered a correctable error. ~~Logging-Measuring~~ the frequency of Link-level correctable errors may be helpful for profiling the integrity of a Link.

Correctable errors also include transaction-level cases where one agent detects an error with a TLP, but another agent is responsible for taking any recovery action if needed, such as re-attempting the operation with a separate subsequent transaction. The detecting agent can be configured to report the error as being correctable since the recovery agent may be able to correct it. If recovery action is indeed needed, the recovery agent must report the error as uncorrectable if the recovery agent decides not to attempt recovery.

### 6.2.2.2 Uncorrectable Errors

Uncorrectable errors are those error conditions that impact functionality of the interface. There is no PCI Express mechanism defined in this specification to correct these errors.

Reporting an uncorrectable error is analogous to asserting SERR# in PCI/PCI-X. For more robust error handling by the system, PCI Express further classifies uncorrectable errors as Fatal and Non-fatal.

#### 6.2.2.2.1 Fatal Errors

Fatal errors are uncorrectable error conditions which render the particular PCI Express Link and related hardware unreliable. For Fatal errors, a reset of the components on the Link may be required to return to reliable operation. Platform handling of Fatal errors, and any efforts to limit the effects of these errors, is platform implementation specific.

Comparing with PCI/PCI-X, reporting a Fatal error is somewhat analogous to asserting SERR#.

#### 6.2.2.2.2 Non-Fatal Errors

Non-fatal errors are uncorrectable errors which cause a particular transaction to be unreliable but the Link is otherwise fully functional. Isolating Non-fatal from Fatal errors provides Requester/Receiver logic in a device or system management software the opportunity to recover from the error without resetting the components on the Link and disturbing other transactions in progress. Devices not associated with the transaction in error are not impacted by the error.

## 6.2.3 Error Signaling

There are ~~two~~three complementary mechanisms in PCI Express which allow the agent detecting an error to alert the system or ~~the initiating another~~ device that an error has

occurred. The first mechanism is through a Completion Status ~~and~~, the second method is with in-band error Messages, and the third is with Error Forwarding (also known as data poisoning).

Note that it is the responsibility of the agent detecting the error to signal the error appropriately.

Section 6.2.6 describes all the errors and how the hardware is required to respond when the error is detected.

### 6.2.3.1 Completion Status

The Completion Status field in the Completion header indicates when the associated Request failed (refer to Section 2.2.9). This is the only one method of error reporting in PCI Express which enables the Requestor to associate an error with a specific Request. In other words, since Non-Posted Requests are not considered complete until after the Completion returns, the Completion Status field gives the ~~initiator~~ Requester an opportunity to “fix” the problem at some higher level protocol (outside the scope of this specification). For example, if a Read is issued to prefetchable memory space and the Completion returns with a Unsupported Request Completion Status, perhaps due to a temporary condition, the ~~initiator~~ Requester may choose to reissue the Read Request without side effects. Note that from a PCI Express point of view, the reissued Read Request is a distinct Request, and there is no relationship (on PCI Express) between the initial Request and the reissued Request.

### 6.2.3.2 Error Messages

...

When multiple errors of the same severity are detected, the corresponding error Messages may be merged for different errors of the same severity. At least one error Message must be sent for detected errors of each severity level. Note, however, that the detection of a given error in some cases will preclude the reporting of certain other errors. See Section 6.2.3.2.3.

*Insert new Implementation Note at the end of Section 6.2.3.2, page 293:*



#### IMPLEMENTATION NOTE

##### Use of ERR\_COR, ERR\_NONFATAL, and ERR\_FATAL

In the 1.0 and 1.0a specifications, a given error was either correctable, non-fatal, or fatal. Assuming signaling was enabled, correctable errors were always signaled with ERR\_COR, non-fatal errors were always signaled with ERR\_NONFATAL, and fatal errors were always signaled with ERR\_FATAL.

In subsequent specifications that support Role-Based Error Reporting, non-fatal errors are sometimes signaled with ERR\_NONFATAL, sometimes signaled with ERR\_COR, and sometimes not signaled at all, depending upon the role of the agent that detects the error and whether the agent implements AER. See Section 6.2.3.2.4. On some platforms, sending

ERR\_NONFATAL will preclude another agent from attempting recovery or determining the ultimate disposition of the error. For cases where the detecting agent is not the appropriate agent to determine the ultimate disposition of the error, a detecting agent with AER can signal the non-fatal error with ERR\_COR, which serves as an advisory notification to software. For cases where the detecting agent is the appropriate one, the agent signals the non-fatal error with ERR\_NONFATAL.

For a given uncorrectable error that's normally non-fatal, if software wishes to avoid continued hierarchy operation upon the detection of that error, software can configure detecting agents that implement AER to escalate the severity of that error to fatal. A detecting agent (if enabled) will always signal a fatal error with ERR\_FATAL, regardless of the agent's role.

Software should recognize that a single transaction can be signaled by multiple agents using different types of error Messages. For example, a poisoned TLP might be signaled by intermediate Receivers with ERR\_COR, while the ultimate destination Receiver might signal it with ERR\_NONFATAL.

---

*Change beginning at Section 6.2.3.2.2, page 294 as follows:*

#### 6.2.3.2.2. Masking Individual Errors Messages

Section 6.2.6 lists all the errors governed by this specification and describes when each of the above error Messages are issued. The transmission of these error Messages by class (correctable, non-fatal, fatal) is enabled~~may be masked~~ using the Reporting Enable fields of the Device Control register (see Section 7.8.4) or the SERR# Enable field in the PCI Command register (see Section 7.5.1.1).

For devices implementing the Advanced Error Reporting capability the Uncorrectable Errors Mask register and Correctable Errors Mask register allows each error condition to be masked independently. If Messages for a particular typeclass of error are ~~blocked~~ not enabled by the combined settings in the Device Control register and PCI Command register, then no Messages of that typeclass will be sent regardless of the values for the corresponding mask register.

~~When~~ If an individual error is masked when it is detected, ~~the its~~ error status bits ~~are~~ is still affected, but no error reporting Message is sent to the Root Complex, and the Header Log and First Error Pointer registers are unmodified.

#### 6.2.3.2.3. Error Pollution

Error pollution can occur if error conditions for a given transaction are not isolated to the error's first most significant occurrence. For example, assume the Physical Layer detects a Receiver Error. This error is detected at the Physical Layer and an error is reported to the Root Complex. To avoid having this error propagate and cause subsequent errors at upper layers (for example, a TLP error at the Data Link Layer), making it more difficult to determine the root cause of the error, subsequent errors which occur for the same packet will not be ~~signaled at~~ reported by the Data Link or Transaction layers. Similarly, when the

Data Link Layer detects an error, subsequent errors which occur for the same packet will not be signaled or reported by the Transaction Layer. This behavior applies only to errors that are associated with a particular packet – other errors are reported for each occurrence.

For errors detected in the Transaction layer, it is permitted and recommended that no more than one error be reported for a single received TLP, and that the following precedence (from highest to lowest) be used:

- Receiver Overflow
- Flow Control Protocol Error
- ECRC Check Failed
- Malformed TLP
- Unsupported Request (UR), Completer Abort (CA), or Unexpected Completion<sup>1</sup>
- Poisoned TLP Received

The Completion Timeout error is not in the above precedence list, since it is not detected by processing a received TLP.

Here's an example of the rationale behind the precedence list. If an ECRC Check fails for a given TLP, the entire contents of the TLP including its header is potentially corrupt, so it makes little sense to report errors like Malformed TLP or Unsupported Request detected with the TLP.

#### 6.2.3.2.4. Advisory Non-Fatal Error Cases

In some cases the detector of a non-fatal error is not the most appropriate agent to determine whether the error is recoverable or not, or if it even needs any recovery action at all. For example, if software attempts to perform a configuration read from a non-existent device, the resulting UR Status in the Completion will signal the error to software, and software does not need for the Completer in addition to signal the error by sending an ERR\_NONFATAL Message. In fact, on some platforms, signaling the error with ERR\_NONFATAL results in a System Error, which breaks normal software probing.

“Advisory Non-Fatal Error” cases are predominantly determined by the role of the detecting agent (Requester, Completer, or Receiver) and the specific error. In such cases, an agent with AER signals the non-fatal error (if enabled) by sending an ERR\_COR Message as an advisory to software, instead of sending ERR\_NONFATAL. An agent without AER sends no Error Message for these cases, since software receiving ERR\_COR would be unable to distinguish Advisory Non-Fatal Error cases from the correctable error cases used to assess link integrity.

Following are the specific cases of Advisory Non-Fatal Errors. Note that multiple errors from the same or different error classes (correctable, non-fatal, fatal) may be present with a single TLP. For example, an unexpected Completion might also be poisoned. See Section 6.2.3.2.3 for requirements and recommendations on reporting multiple errors. For the previous example, it is recommended that “Unexpected Completion” be reported, and that “Poisoned TLP Received” not be reported.

---

<sup>1</sup> These are mutually exclusive errors, so their relative order does not matter.

If software wishes for an agent with AER to handle what would normally be an Advisory Non-Fatal Error case as being more serious, software can escalate the severity of the uncorrectable error to fatal, in which case the agent (if enabled) will signal the error with ERR\_FATAL.

#### 6.2.3.2.4.1. Completer Sending a Completion with UR/CA Status

A Completer generally sends a Completion with an Unsupported Request or Completer Abort (UR/CA) Status to signal a uncorrectable error for a Non-Posted Request<sup>2</sup>. If the severity of the UR/CA error is non-fatal, the Completer must handle this case as an Advisory Non-Fatal Error<sup>3</sup>. A Completer with AER signals the non-fatal error (if enabled) by sending an ERR\_COR Message. A Completer without AER sends no Error Message for this case.

Even though there was an uncorrectable error for this specific transaction, the Completer must handle this case as an Advisory Non-Fatal Error, since the Requester upon receiving the Completion with UR/CA Status is responsible for reporting the error (if necessary) using a Requester-specific mechanism (see Section 6.2.3.2.5).

#### 6.2.3.2.4.2. Intermediate Receiver

When a Receiver that's not serving as the ultimate PCIe destination for a TLP detects<sup>4</sup> a non-fatal error with the TLP, this "intermediate" Receiver must handle this case as an Advisory Non-Fatal Error<sup>5</sup>. A Receiver with AER signals the error (if enabled) by sending an ERR\_COR Message. A Receiver without AER sends no Error Message for this case. An exception to the intermediate Receiver case for Root Complexes (RCs) and Bridges is noted below.

An example where the intermediate Receiver case occurs is a Switch that detects poison or bad ECRC in a TLP that it is routing. Even though this was an uncorrectable (but non-fatal) error at this point in the TLP's route, the intermediate Receiver handles it as an Advisory Non-Fatal Error, so that the ultimate Receiver of the TLP (i.e., the Completer for a Request TLP, or the Requester for a Completion TLP) is not precluded from handling the error more appropriately according to its error settings. For example, a given Completer that detects poison in a Memory Write Request<sup>6</sup> might have the error masked (and thus go

---

<sup>2</sup> If the Completer is returning data in a Completion, and the data is bad or suspect, the Completer is permitted to signal the error using the Error Forwarding (Data Poisoning) mechanism instead of handling it as a UR or CA.

<sup>3</sup> If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR\_FATAL.

<sup>4</sup> If the Receiver does not implement ECRC Checking or ECRC Checking is not enabled, the Receiver will not detect an ECRC Check Failed error.

<sup>5</sup> If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR\_FATAL.

<sup>6</sup> See Section 2.7.2.2 for special rules that apply for poisoned Memory Write Requests.

unsigned), whereas a different Completer in the same hierarchy might signal that error with ERR\_NONFATAL.

If an RC detects a non-fatal error with a TLP it normally would forward peer-to-peer between Root Ports, but the RC does not support propagating the error related information (e.g., a TLP Digest, EP bit, or equivalent) with the forwarded transaction, the RC must signal the error (if enabled) with ERR\_NONFATAL and also must not forward the transaction. An example is an RC needing to forward a poisoned TLP peer-to-peer between Root Ports, but the RC's internal fabric does not support poison indication.

#### 6.2.3.2.4.3. Ultimate PCIe Receiver of a Poisoned TLP

When a poisoned TLP is received by its ultimate PCIe destination, if the severity is non-fatal and the Receiver deals with the poisoned data in a manner that permits continued operation, the Receiver must handle this case as an Advisory Non-Fatal Error<sup>7</sup>. A Receiver with AER signals the error (if enabled) by sending an ERR\_COR Message. A Receiver without AER sends no Error Message for this case. See Section 2.7.2.2 for special rules that apply for poisoned Memory Write Requests.

An example is a Root Complex that receives a poisoned Memory Write TLP that targets host memory. If the Root Complex propagates the poisoned data along with its indication to host memory, it signals the error (if enabled) with an ERR\_COR. If the Root Complex does not propagate the poison to host memory, it signals the error (if enabled) with ERR\_NONFATAL.

Another example is a Requester that receives a poisoned Memory Read Completion TLP. If the Requester propagates the poisoned data internally or handles the error like it would for a Completion with UR/CA Status, it signals the error (if enabled) with an ERR\_COR. If the Requester does not handle the poison in a manner that permits continued operation, it signals the error (if enabled) with ERR\_NONFATAL.

#### 6.2.3.2.4.4. Requester with Completion Timeout

When the Requester of a Non-Posted Request times out while waiting for the associated Completion, the Requester is permitted to attempt to recover from the error by issuing a separate subsequent Request. The Requester is permitted to attempt recovery zero, one, or multiple (finite) times, but must signal the error (if enabled) with an uncorrectable error Message if no further recovery attempt will be made.

If the severity of the Completion Timeout is non-fatal, and the Requester elects to attempt recovery by issuing a new request, the Requester must first handle the current error case as an Advisory Non-Fatal Error<sup>8</sup>. A Requester with AER signals the error (if enabled) by sending an ERR\_COR Message. A Requester without AER sends no Error Message for this case.

---

<sup>7</sup> If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR\_FATAL.

<sup>8</sup> If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR\_FATAL. The Requester is strongly discouraged from attempting recovery since sending ERR\_FATAL will often result in the entire hierarchy going down.

Note that automatic recovery by the Requester from a Completion Timeout is generally possible only if the Non-Posted Request has no side-effects, but may also depend upon other considerations outside the scope of this specification.

#### 6.2.3.2.4.5. Receiver of an Unexpected Completion

When a Receiver receives an unexpected Completion and the severity of the Unexpected Completion error is non-fatal, the Receiver must handle this case as an Advisory Non-Fatal Error<sup>9</sup>. A Receiver with AER signals the error (if enabled) by sending an ERR\_COR Message. A Receiver without AER sends no Error Message for this case.

If the unexpected Completion was a result of misrouting, the Completion Timeout mechanism at the associated Requester will trigger eventually, and the Requester may elect to attempt recovery. Interference with Requester recovery can be avoided by having the Receiver of the unexpected Completion handle the error as an Advisory Non-Fatal Error.

#### 6.2.3.2.5. Requester Receiving a Completion with UR/CA Status

When a Requester receives back a Completion with a UR/CA Status, generally the Completer has handled the error as an Advisory Non-Fatal Error, assuming the error severity was non-fatal at the Completer (see Section 6.2.3.2.4.1). The Requester must determine if any error recovery action is necessary, what type of recovery action to take, and whether or not to report the error.

If the Requester needs to report the error, the Requester must do so solely through a Requester-specific mechanism. For example, many devices have an associated device driver that can report errors to software. As another important example, the Root Complex on some platforms returns all 1's to software if a Configuration Read Completion has a UR/CA Status.

The Requester is not permitted to report the error using PCIe logging and error Message signaling.

#### 6.2.3.3. Error Forwarding (Data Poisoning)

Error Forwarding, also known as data poisoning, is indicated by setting the EP field in a TLP. See Section 2.7.2. This is another method of error reporting in PCI Express that enables the Receiver of a TLP to associate an error with a specific Request or Completion. In contrast to the Completion Status mechanism, however, Error Forwarding can be used with either Requests or Completions that contain data. In addition, “intermediate” Receivers along the TLP’s route, not just the Receiver at the ultimate destination, are required to detect and report (if enabled) receiving the poisoned TLP. This can help software determine if a particular Switch along the path poisoned the TLP.

---

<sup>9</sup> If the severity is fatal, the error is not an Advisory Non-Fatal Error, and must be signaled (if enabled) with ERR\_FATAL.

*Insert the following prior to Section 6.2.5, page 301:*

### 6.2.4.3. Advisory Non-Fatal Error Logging

Section 6.2.3.2.4 describes Advisory Non-Fatal Error cases, under which an agent with AER detecting an uncorrectable error of non-fatal severity signals the error (if enabled) using ERR\_COR instead of ERR\_NONFATAL. For the same cases, an agent without AER sends no Error Message. The remaining discussion in this section is in the context of agents that do implement AER.

For Advisory Non-Fatal Error cases, since an uncorrectable error is signaled using the correctable Error Message, control/status/mask bits involving both uncorrectable and correctable errors apply. Figure 6-3 shows a flowchart of the sequence. Following are some of the unique aspects for logging Advisory Non-Fatal Errors.

First, the uncorrectable error needs to be of severity non-fatal, as determined by the associated bit in the Uncorrectable Error Severity register. If the severity is fatal, the error does not qualify as an Advisory Non-Fatal Error, and will be signaled (if enabled) with ERR\_FATAL.

Next, the specific error case needs to be one of the Advisory Non-Fatal Error cases documented in Section 6.2.3.2.4. If not, the error does not qualify as an Advisory Non-Fatal Error, and will be signaled (if enabled) with an uncorrectable Error Message.

Next, the Advisory Non-Fatal Error Status bit is set in the Correctable Error Status register to indicate the occurrence of the advisory error, and the Advisory Non-Fatal Error Mask bit in the Correctable Error Mask register is checked to determine whether to proceed further with logging and signaling.

If the Advisory Non-Fatal Error Mask bit is clear, logging proceeds by setting the “corresponding” bit in the Uncorrectable Error Status register, based upon the specific uncorrectable error that’s being reported as an advisory error. If the “corresponding” uncorrectable error bit in the Uncorrectable Error Mask register is clear, the First Error Pointer and Header Log registers are updated to log the error, assuming they are not still “occupied” by a previous unserved error.

Finally, an ERR\_COR Message is sent if the Correctable Error Reporting Enable bit is set in the Device Control register.

Replace Figure 6-3, page 298 with the following:

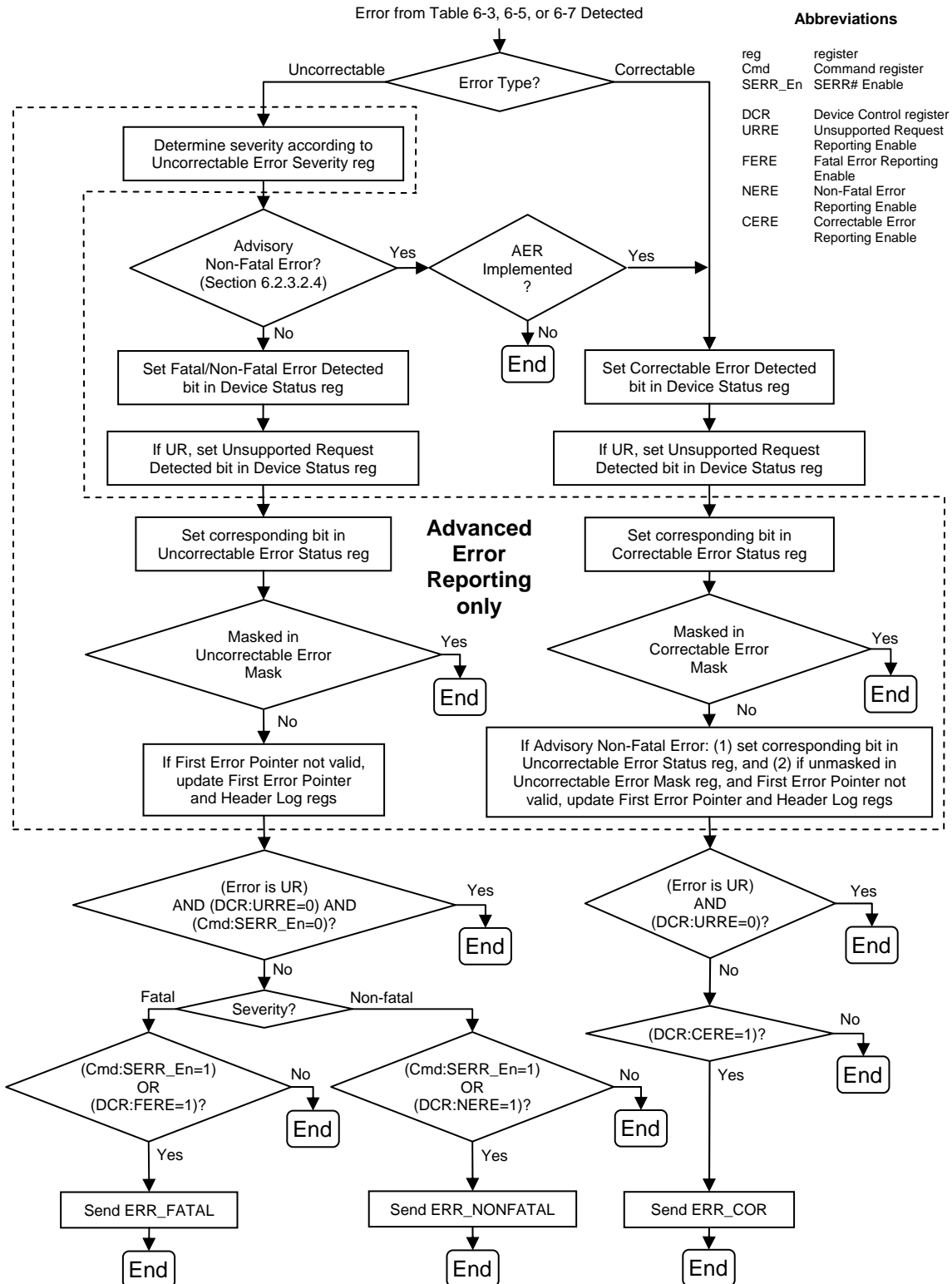


Figure 6-3: Flowchart Showing Sequence of Device Error Signaling and Logging Operations

Insert new section prior to Section 6.2.6, page 299:

## 6.2.x. Error Message Controls

Error Messages have a complex set of associated control and status bits. Figure 6-xx provides a conceptual summary in the form of a pseudo logic diagram for how error Messages are generated, logged, forwarded, and ultimately notified to the system. Not all logged status bits are shown. The logic gates shown in this diagram are intended for conveying general concepts, and not for direct implementation.

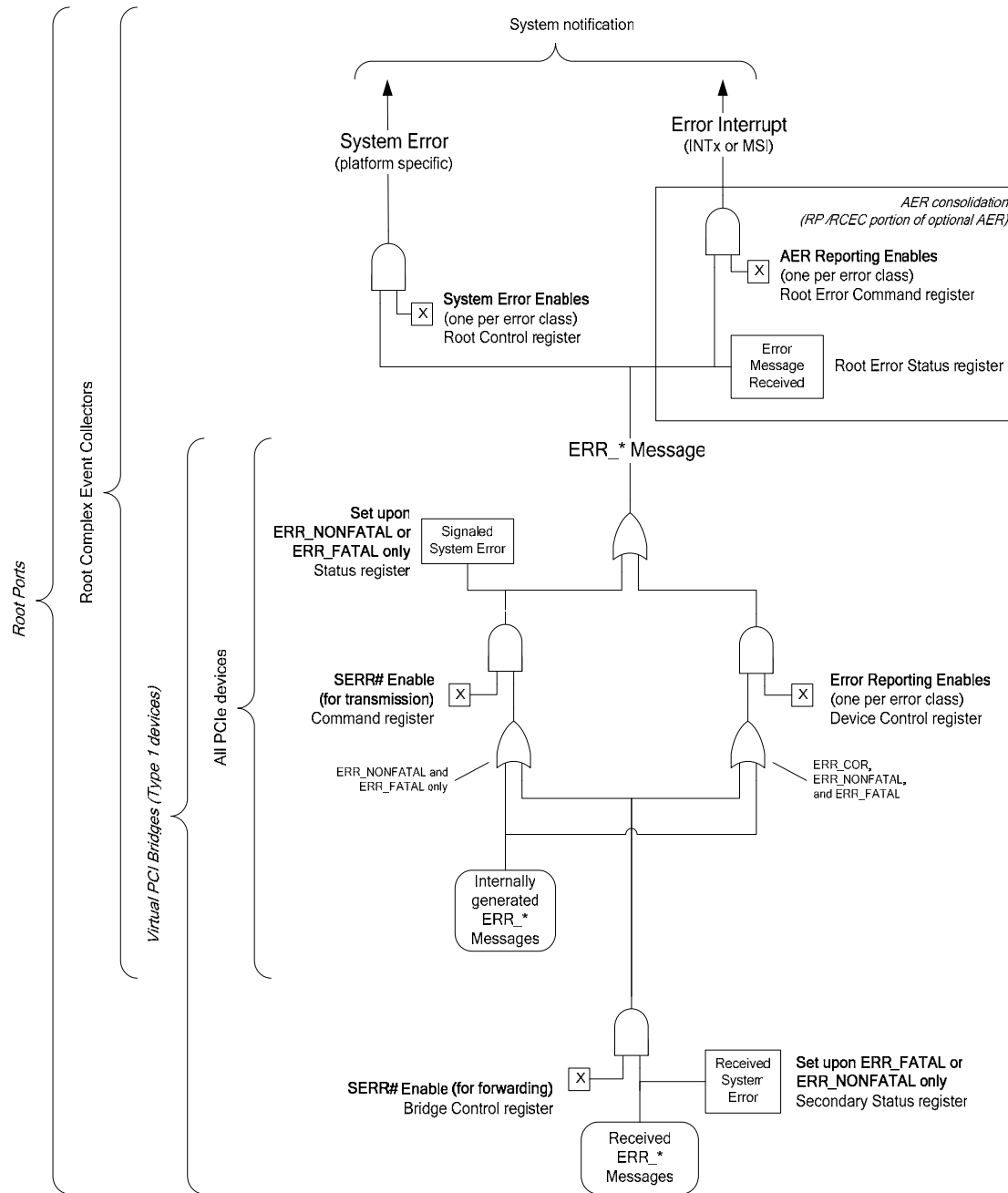


Figure 6-xx: Pseudo Logic Diagram for Error Message Controls

Change Section 6.2.6, page 298 as follows:  
 (Note: this section already incorporates 1.0a ECN/Errata text.)

## 6.2.6. Error Listing and Rules

...

Table 6-3: Physical Layer Error List

Error Name	<a href="#">Error Type</a> <a href="#">Default Severity</a>	Detecting Agent Action	References
------------	--	------------------------	------------

...

Table 6-5: Data Link Layer Error List

Error Name	<a href="#">Error Type</a> <a href="#">(Default Severity)</a>	Detecting Agent Action	References
------------	--	------------------------	------------

...

Table 6-7: Transaction Layer Error List

Error Name	<a href="#">Error Type</a> <a href="#">(Default Severity)</a>	Detecting Agent Action	References
Poisoned TLP Received	Uncorrectable (Non-Fatal)	<i>Receiver (if data poisoning is supported):</i> Send ERR_NONFATAL to Root Complex, <u>or</u> <a href="#">ERR_COR for the Advisory Non-Fatal Error cases described in Sections 6.2.3.2.4.2 and 6.2.3.2.4.3.</a> Log the header of the Poisoned TLP.	<same as 1.1RD>
ECRC Check Failed		<i>Receiver (if ECRC checking is supported):</i> Send ERR_NONFATAL to Root Complex, <u>or</u> <a href="#">ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.2.</a> Log the header of the TLP that encountered <u>ed</u> the ECRC error.	<same as 1.1RD>

Error Name	<u>Error Type</u> (Default Severity)	Detecting Agent Action	References
Unsupported Request (UR)		<p><i>Request Receiver:</i></p> <p>Send ERR_NONFATAL to Root Complex, <u>or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1.</u></p> <p>Log the header of the TLP that caused the error.</p>	<same as 1.1RD>
Completion Timeout		<p><i>Requester:</i></p> <p>Send ERR_NONFATAL to Root Complex, <u>or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.4.</u></p>	<same as 1.1RD>
Completer Abort		<p><i>Completer:</i></p> <p>Send ERR_NONFATAL to Root Complex, <u>or ERR_COR for the Advisory Non-Fatal Error case described in Section 6.2.3.2.4.1.</u></p> <p>Log the header of the Request that encountered the error.</p>	<same as 1.1RD>
Unexpected Completion		<p><i>Receiver:</i></p> <p>Send <del>ERR_NONFATAL</del> <u>ERR_COR</u> to Root Complex. <u>This is an Advisory Non-Fatal Error case described in Section 6.2.3.2.4.5.</u></p> <p>Log the header of the Completion that encountered the error.</p> <p><del>Note that if the Unexpected Completion is a result of misrouting, the Completion Timeout mechanism will be triggered at the corresponding Requester.</del></p>	<same as 1.1RD>

For all errors listed above, the appropriate status bit(s) must be set upon detection of the error. For Unsupported Request (UR), additional detection and reporting enable bits apply. See Section 6.2.5. the logging and signaling control/status mechanisms are more complicated than for the other types of errors, and so are clarified below. Note that upon receiving a Non-Posted (NP) Request that is UR, the return of a Completion with UR status is unconditional, and the reporting settings in the Device Control (and Uncorrectable Error

~~Mask register if Advanced Error Reporting (AER) is implemented) register(s) do not affect the return of this completion.~~

~~In the Device Status register:~~

- ~~Set Unsupported Request Detected any time a received Request is determined to be UR.~~
- ~~Set Non-Fatal Error Detected any time a received Request is determined to be UR, and either AER is not implemented or AER Unsupported Request Error Severity is set to Non-Fatal.~~
- ~~Set Fatal Error Detected any time a received Request is determined to be UR, and AER is implemented, and AER Unsupported Request Error Severity is set to Fatal.~~

~~UR is also reported through the AER structure if implemented.~~

~~If the Unsupported Request Reporting Enable bit in the Device Control register is not set, the device must not send an error Message in the case of a UR, although the device will set status bits as described above (regardless of AER settings if AER is implemented).~~

~~If AER is not implemented, and if the Unsupported Request Reporting Enable bit in the Device Control register is set, an ERR\_NONFATAL error Message will be sent to the Root Complex (or signaled within the Root Complex, in the case of a Root Port) on detection of a UR.~~

~~If AER is implemented and the Unsupported Request Reporting Enable bit in the Device Control register is set, an error Message is sent if the Unsupported Request Error Mask bit is not set. The severity of the error Message is determined by the setting of the Unsupported Request Error Severity bit.~~



## IMPLEMENTATION NOTE

### Device UR Reporting Compatibility with Legacy and 1.0a Software

With 1.0a devices that do not implement Role-Based Error Reporting<sup>10</sup>, the Unsupported Request Reporting Enable bit in the Device Control register, when clear, prevents the device from sending any error Message to signal a UR error. With Role-Based Error Reporting devices, if the SERR# Enable bit in the Command register is set, the device is implicitly enabled<sup>11</sup> to send ERR\_NONFATAL or ERR\_FATAL messages to signal UR errors, even if the Unsupported Request Reporting Enable bit is clear. This raises a backward compatibility concern with software (or firmware) written for 1.0a devices.

With software/firmware that sets the SERR# Enable bit but leaves the Unsupported Request Reporting Enable and Correctable Error Reporting Enable bits clear, a Role-Based

---

<sup>10</sup> As indicated by the Role-Based Error Reporting bit in the Device Capabilities register. See Section 7.8.3.

<sup>11</sup> Assuming the Unsupported Request Error Mask bit is not set in the Uncorrectable Error Mask register if the device implements AER.

Error Reporting device that encounters a UR error will send no error Message if the Request was non-posted, and will signal the error with ERR\_NONFATAL if the Request was posted. The behavior with non-posted Requests supports PC-compatible configuration space probing, while the behavior with posted Requests restores error reporting compatibility with PCI and PCI-X, avoiding the potential in this area for silent data corruption. Thus, Role-Based Error Reporting devices are backward compatible with envisioned legacy and 1.0a software and firmware.

---

Change Section 6.2.7.1, page 303 as follows:

(Note: this section already incorporates 1.0a ECN/Errata text.)

### 6.2.7.1 Error Message Forwarding and PCI Mapping for Bridge - Rules

In general, a TLP is either passed from one side of the Virtual PCI Bridge to the other, or is handled at the ingress side of the Bridge according to the same rules which apply to the ultimate recipient of a TLP. The following rules cover PCI Express specific error related cases: See Section 6.2.x for a conceptual summary on Error Message Controls.

...

- ❑ ~~ERR\_COR, ERR\_NONFATAL, and ERR\_FATAL are forwarded from the secondary interface to the primary interface, if the SERR# Enable bit in the Bridge Control register is set. A Bridge forwarding an error Message must not set the corresponding Error Detected bit in the Device Status register. ~~Transmission of forwarded ERR\_NONFATAL and ERR\_FATAL messages by the primary interface is controlled enabled by the SERR# Enable bit in the Command register. Alternatively (or in addition), transmission of forwarded ERR\_COR, ERR\_NONFATAL, and ERR\_FATAL messages by the primary interface is enabled by their respective Error Reporting Enable bits in the Device Control register.~~ Transmission of forwarded error Messages by the primary interface is controlled by multiple bits, as shown in Figure 6-xx. <Pseudo Logic Diagram for Error Message Controls>~~
- ❑ For a Root Port, error Messages forwarded from the secondary interface to the primary interface must be enabled for “transmission” by the primary interface in order to cause a System Error via the Root Control register or (when the Advanced Error Reporting Capability is present) reporting via the Root Error Command Register and logging in the Root Error Status Register and Error Source Identification Register
- ❑ For a Root Complex Event Collector (technically not a Bridge), error Messages “received” from associated Root Complex Integrated Endpoints must be enabled for “transmission” in order to cause a System Error via the Root Control register or (when the Advanced Error Reporting Capability is present) reporting via the Root Error Command Register and logging in the Root Error Status Register and Error Source Identification Register. ~~Since a Root Complex Event Collector has no Bridge Control register, ERR\_COR is always enabled for “transmission”, while ERR\_NONFATAL and ERR\_FATAL are enabled for “transmission” by the SERR# Enable bit in the Command register.~~

Change Section 7.8.3, page 381 as follows:

### 7.8.3. Device Capabilities Register (Offset 04h)

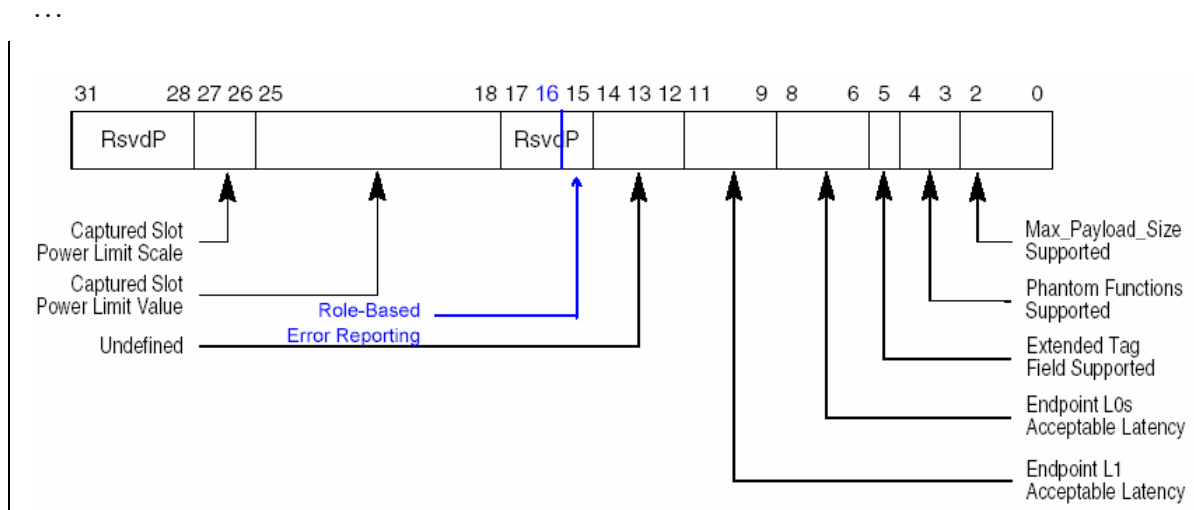


Figure 7-24: Device Capabilities Register

Table 7-21: Device Capabilities Register

Bit Location	Register Description	Attributes
...	...	...
15	<a href="#">Role-Based Error Reporting</a> – this bit, when set, indicates that the device implements the functionality originally defined in the <a href="#">Error Reporting ECN for PCIe Base Specification 1.0a</a> , and later incorporated into <a href="#">PCIe Base Specification 1.1</a> . This bit must be set by all devices conforming to the ECN, <a href="#">PCIe Base 1.1</a> , or subsequent <a href="#">PCIe Base</a> revisions.	RO

Change Table 7-23, page 387 as follows:

Table 7-23: Device Control Register

Bit Location	Register Description	Attributes
0	<b>Correctable Error Reporting Enable</b> – This bit <a href="#">in conjunction with other bits</a> controls <del>reporting of correctable errors</del> <a href="#">sending ERR_COR Messages</a> . Refer to Sections <a href="#">6.2.5</a> and <a href="#">6.2.x</a> for <del>further</del> details. For a multi-function device, this bit controls error reporting for each function from point-of-view of the respective function.	RW
...	...	...

Bit Location	Register Description	Attributes
1	<p><b>Non-Fatal Error Reporting Enable</b> – This bit <a href="#">in conjunction with other bits</a> controls <a href="#">reporting of Non-fatal errors</a> sending <a href="#">ERR_NONFATAL Messages</a>. Refer to Sections <a href="#">6.2.5</a> and <a href="#">6.2.x</a> for <del>further</del> details. For a multi-function device, this bit controls error reporting for each function from point-of-view of the respective function.</p> <p>...</p>	RW
2	<p><b>Fatal Error Reporting Enable</b> – This bit <a href="#">in conjunction with other bits</a> controls <a href="#">reporting of Fatal errors</a> sending <a href="#">ERR_FATAL Messages</a>. Refer to Sections <a href="#">6.2.5</a> and <a href="#">6.2.x</a> for <del>further</del> details. For a multi-function device, this bit controls error reporting for each function from point-of-view of the respective function.</p> <p>...</p>	RW
3	<p><b>Unsupported Request Reporting Enable</b> – This bit <a href="#">in conjunction with other bits</a> <del>enables</del> controls <a href="#">reporting the signaling</a> of Unsupported Requests <a href="#">by sending Error Messages when set</a>. Refer to Sections <a href="#">6.2.5</a> and <a href="#">6.2.x</a> for <del>further</del> details. For a multi-function device, this bit controls error reporting for each function from point-of-view of the respective function. <del>Note that the reporting of error Messages (ERR_COR, ERR_NONFATAL, ERR_FATAL) received by Root Port is controlled exclusively by Root Control register described in Section 7.8.12.</del></p> <p>...</p>	RW

*Insert new Implementation Note following the end of Table 7-23, page 390:*



## IMPLEMENTATION NOTE

### Software UR Reporting Compatibility with 1.0a Devices

With 1.0a devices<sup>12</sup>, if the Unsupported Request Reporting Enable bit is set, the device when operating as a Completer will send an uncorrectable error Message (if enabled) when a UR error is detected. On platforms where an uncorrectable error Message is handled as a System Error, this will break PC-compatible configuration space probing, so software/firmware on such platforms may need to avoid setting the Unsupported Request Reporting Enable bit.

With devices implementing Role-Based Error Reporting, setting the Unsupported Request Reporting Enable bit will not interfere with PC-compatible configuration space probing, assuming that the severity for UR is left at its default of non-fatal. However, setting the

<sup>12</sup> In this context, “1.0a devices” are devices that do not implement Role-Based Error Reporting.

[Unsupported Request Reporting Enable bit will enable the device to report UR errors<sup>13</sup> detected with posted Requests, helping avoid this case for potential silent data corruption.](#)

[On platforms where robust error handling and PC-compatible configuration space probing is required, it is suggested that software or firmware have the Unsupported Request Reporting Enable bit set for Role-Based Error Reporting devices, but clear for 1.0a devices. Software or firmware can distinguish the two classes of devices by examining the Role-Based Error Reporting bit in the Device Capabilities register.](#)

---

Change Section 7.10.2, page 423 as follows:

## 7.10.2 Uncorrectable Error Status Register (Offset 04h)

The Uncorrectable Error Status register [reports-indicates](#) error [detection](#) status of individual errors [sources](#) on a PCI Express device. An individual error status bit that is set indicates that a particular error [occurred was detected](#); software may clear an error status by writing a 1 to the respective bit. Refer to Section 6.2 for further details. ...

Change Section 7.10.5, page 426 as follows:

## 7.10.5 Correctable Error Status Register (Offset 10h)

...

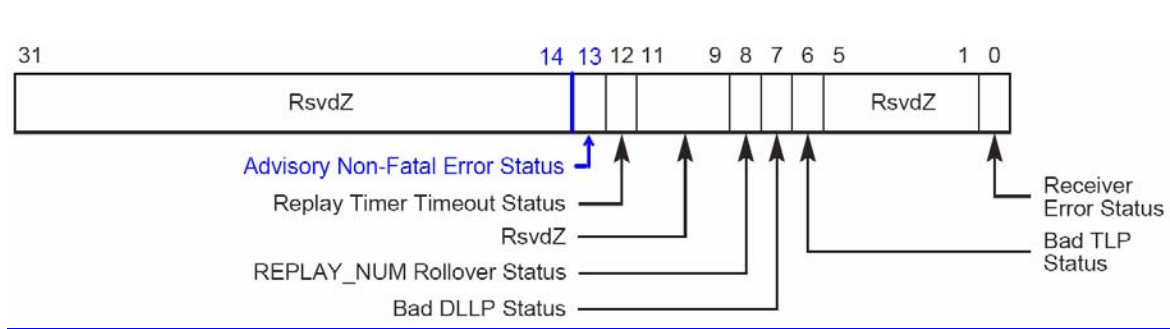


Figure 7-61: Correctable Error Status Register

---

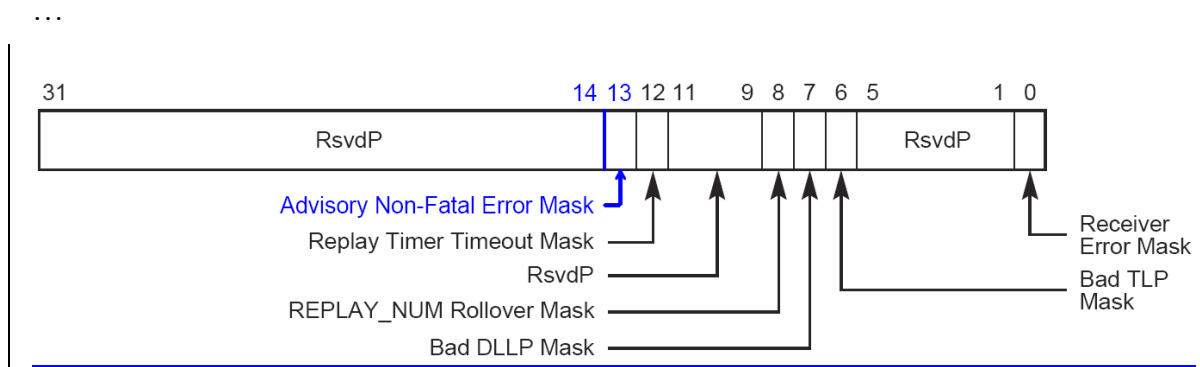
<sup>13</sup> [With Role-Based Error Reporting devices, setting the SERR# Enable bit in the Command register also implicitly enables UR reporting.](#)

**Table 7-54: Correctable Error Status Register**

Bit Location	Register Description	Attributes	Default Value
...	...	...	...
<a href="#">13</a>	<a href="#">Advisory Non-Fatal Error Status</a>	<a href="#">RW1CS</a>	<a href="#">0</a>

Change Section 7.10.6, page 427 as follows:

### 7.10.6 Correctable Error Mask Register (Offset 14h)



**Figure 7-63: Correctable Error Mask Register**

**Table 7-56: Correctable Error Mask Register**

Bit Location	Register Description	Attributes	Default Value
...	...	...	...
<a href="#">13</a>	<a href="#">Advisory Non-Fatal Error Mask – this bit is set by default to enable compatibility with software that does not comprehend Role-Based Error Reporting.</a>	<a href="#">RWS</a>	<a href="#">1</a>