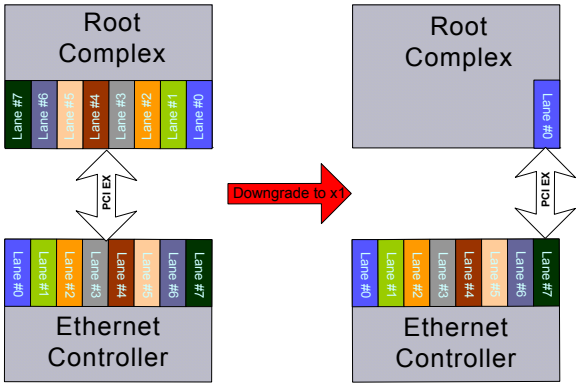


Figure 3-3 Lane Downshift in a Reversal x8 Configuration

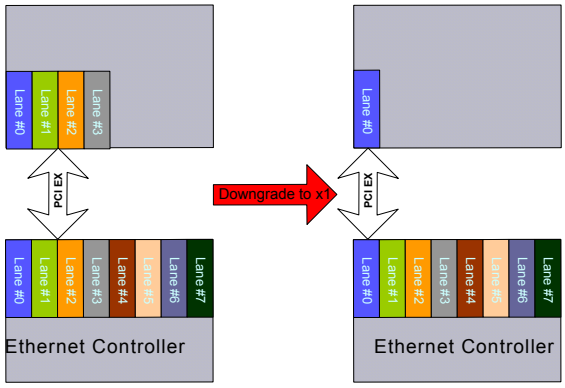


Figure 3-4 Lane Downshift in a x4 Configuration

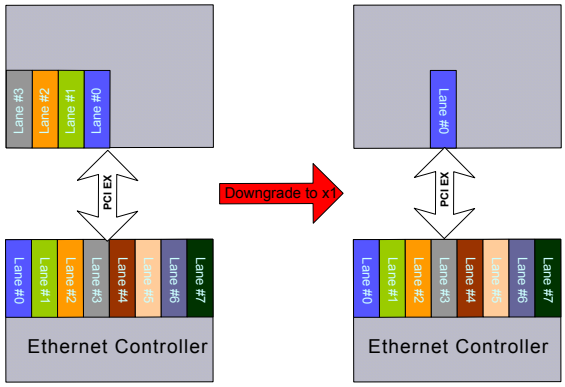


Figure 3-5 Lane Downshift in an x4 Reversal Configuration

3.1.6.7 Reset

The PCIe PHY supplies the core reset to the 82599. The reset can be caused by the following events:

- Upstream move to hot reset — Inband Mechanism (LTSSM).
- Recovery failure (LTSSM returns to detect)
- Upstream component moves to disable.

3.1.6.8 Scrambler Disable

The scrambler/de-scrambler functionality in the 82599 can be eliminated by two mechanisms:

- Upstream according to the PCIe specification
- EEPROM bit — Scram_dis.

3.1.7 Error Events and Error Reporting

3.1.7.1 General Description

PCIe defines two error reporting paradigms: the baseline capability and the Advanced Error Reporting (AER) capability. The baseline error reporting capabilities are required of all PCIe devices and define the minimum error reporting requirements. The AER capability is defined for more robust error reporting and is implemented with a specific PCIe capability structure. Both mechanisms are supported by the 82599.

The *SERR# Enable* and the *Parity Error* bits from the Legacy Command register also take part in the error reporting and logging mechanism.

In a multi-function device, PCIe errors that are not related to any specific function within the device are logged in the corresponding status and logging registers of all functions in that device. These include the following cases of Unsupported Request (UR):

- A memory or I/O access that does not match any BAR for any function
- Messages
- Configuration accesses to a non-existent function

Figure 3-6 shows, in detail, the flow of error reporting in the 82599.

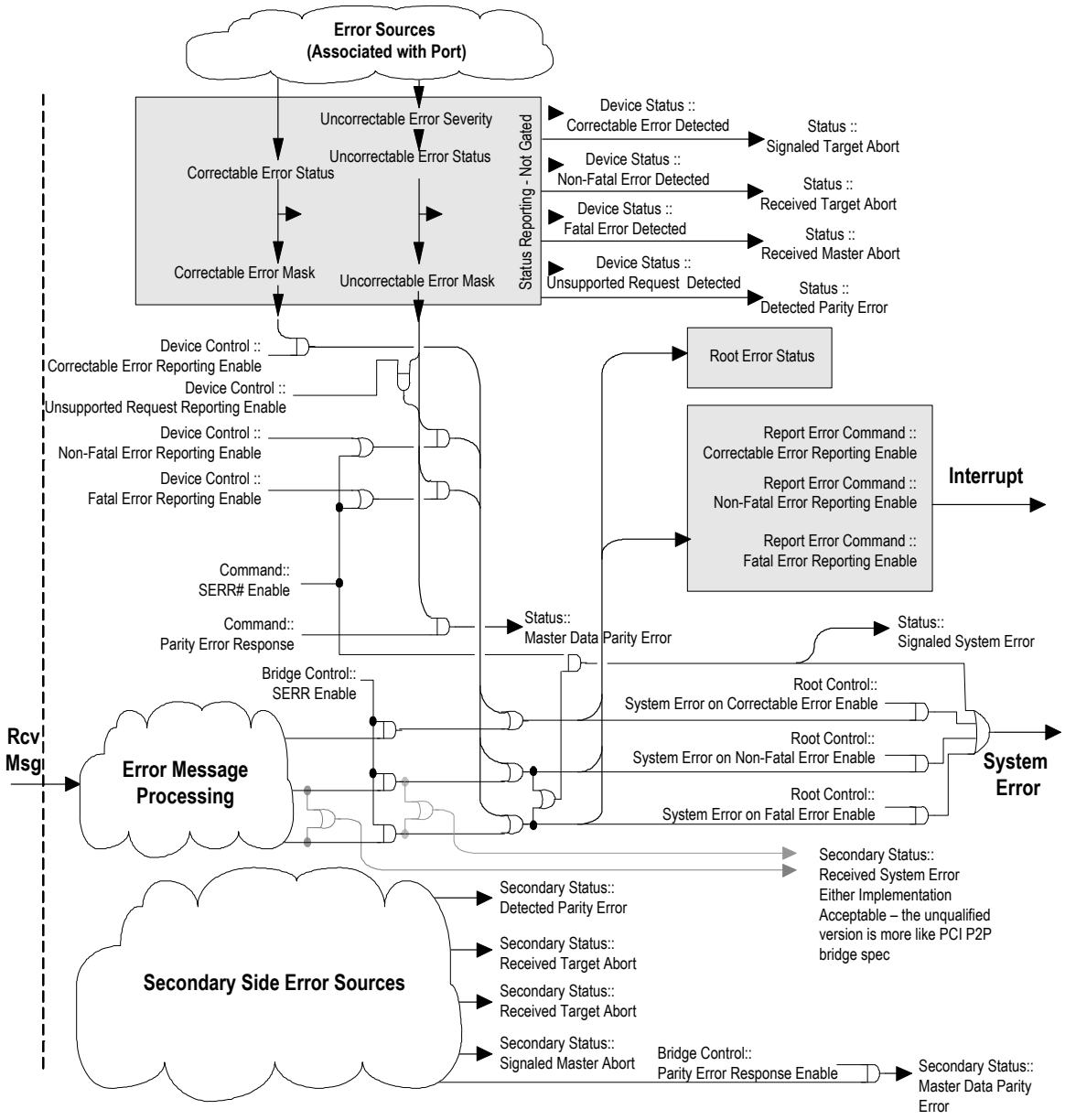


Figure 3-6 Error Reporting Mechanism

3.1.7.2 Error Events

Table 3-8 lists the error events identified by the 82599 and the response in terms of logging, reporting, and actions taken. Refer to the PCIe specification for the effect on the PCI Status register.

Table 3-8 Response and Reporting of PCIe Error Events

Error Name	Error Events	Default Severity	Action
Physical Layer Errors			
Receiver Error	• 8b/10b Decode Errors • Packet Framing Error	Correctable Send ERR_CORR	TLP to Initiate NAK, Drop Data DLLP to Drop
Data Link Errors			
Bad TLP	• Bad CRC • Not Legal EDB • Wrong Sequence Number	Correctable Send ERR_CORR	TLP to Initiate NAK, Drop Data
Bad DLLP	• Bad CRC	Correctable Send ERR_CORR	DLLP το Drop
Replay Timer Timeout	• REPLAY_TIMER expiration	Correctable Send ERR_CORR	Follow LL Rules
REPLAY NUM Rollover	• REPLAY NUM Rollover	Correctable Send ERR_CORR	Follow LL Rules
Data Link Layer Protocol Error	• Violations of Flow Control Initialization Protocol	Uncorrectable Send ERR_FATAL	
TLP Errors			
Poisoned TLP Received	• TLP With Error Forwarding	Uncorrectable ERR_NONFATAL Log Header	If completion TLP: Error is non-fatal (default case) • Send error message if advisory • Retry the request once and send advisory error message on each failure • If fails, send uncorrectable error message Error is defined as fatal • Send uncorrectable error message
Unsupported Request (UR)	• Wrong Config Access • MRdLk • Config Request Type1 • Unsupported Vendor Defined Type 0 Message • Not Valid MSG Code • Not Supported TLP Type • Wrong Function Number • Received TLP Outside Address Range	Uncorrectable ERR_NONFATAL Log header	Send Completion With UR

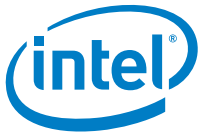


Table 3-8 Response and Reporting of PCIe Error Events (Continued)

Error Name	Error Events	Default Severity	Action
Completion Timeout	• Completion Timeout Timer Expired	Uncorrectable ERR_NONFATAL	Error is non-fatal (default case) • Send error message if advisory • Retry the request once and send advisory error message on each failure • If fails, send uncorrectable error message Error is defined as fatal • Send uncorrectable error message
Completer Abort	• Received Target Access With Data Size >64 bits	Uncorrectable. ERR_NONFATAL Log header	Send completion with CA
Unexpected Completion	• Received Completion Without a Request For It (Tag, ID, etc.)	Uncorrectable ERR_NONFATAL Log Header	Discard TLP
Receiver Overflow	• Received TLP Beyond Allocated Credits	Uncorrectable ERR_FATAL	Receiver Behavior is Undefined
Flow Control Protocol Error	• Minimum Initial Flow Control Advertisements • Flow Control Update for Infinite Credit Advertisement	Uncorrectable. ERR_FATAL	Receiver Behavior is Undefined
Malformed TLP (MP)	• Data Payload Exceed Max_Payload_Size • Received TLP Data Size Does Not Match Length Field • TD field value does not correspond with the observed size • PM Messages That Don't Use TC0. • Usage of Unsupported VC	Uncorrectable ERR_FATAL Log Header	Drop the Packet, Free FC Credits
Completion with Unsuccessful Completion Status		No Action (already done by originator of completion)	Free FC Credits

3.1.7.3 Error Forwarding (TLP Poisoning)

If a TLP is received with an error-forwarding trailer, the packet is dropped and is not delivered to its destination. The 82599 then reacts as described in Table 3-8.

The 82599 does not initiate any additional master requests for that PCI function until it detects an internal software reset for the associated LAN port. Software is able to access device registers after such a fault.

System logic is expected to trigger a system-level interrupt to inform the operating system of the problem. Operating systems can then stop the process associated with the transaction, re-allocate memory to a different area instead of the faulty area, etc.

3.1.7.4 End-to-End CRC (ECRC)

The 82599 supports ECRC as defined in the PCIe specification. The following functionality is provided:

- Inserting ECRC in all transmitted TLPs:
  - The 82599 indicates support for inserting ECRC in the *ECRC Generation Capable* bit of the PCIe configuration registers. This bit is loaded from the *ECRC Generation* EEPROM bit.
  - Inserting ECRC is enabled by the *ECRC Generation Enable* bit of the PCIe configuration registers.
- ECRC is checked on all incoming TLPs. A packet received with an ECRC error is dropped. Note that for completions, a completion timeout occurs later (if enabled), which results in re-issuing the request.
  - The 82599 indicates support for ECRC checking in the *ECRC Check Capable* bit of the PCIe configuration registers. This bit is loaded from the *ECRC Check* EEPROM bit.
  - Checking of ECRC is enabled by the *ECRC Check Enable* bit of the PCIe configuration registers.
- ECRC errors are reported
- System software can configure ECRC independently per each LAN function

3.1.7.5 Partial Read and Write Requests

Partial memory accesses

The 82599 has limited support of read and write requests with only part of the byte enable bits set:

- Partial writes with at least one byte enabled are silently dropped.
- Zero-length writes have no internal impact (nothing written, no effect such as clear-by-write). The transaction is treated as a successful operation (no error event).
- Partial reads with at least one byte enabled are handled as a full read. Any side effect of the full read (such as clear by read) is also applicable to partial reads.
- Zero-length reads generate a completion, but the register is not accessed and undefined data is returned.

Note: The 82599 does not generate an error indication in response to any of the previous events.

Partial I/O accesses

- Partial access on address
  - A write access is discarded
  - A read access returns 0xFFFF

- Partial access on data, where the address access was correct
  - A write access is discarded
  - A read access performs the read

3.1.7.6 Error Pollution

Error pollution can occur if error conditions for a given transaction are not isolated to the error's first occurrence. If the PHY detects and reports a receiver error, to avoid having this error propagate and cause subsequent errors at the upper layers, the same packet is not signaled at the data link or transaction layers. Similarly, when the data link layer detects an error, subsequent errors that occur for the same packet are not signaled at the transaction layer.

3.1.7.7 Completion With Unsuccessful Completion Status

A completion with unsuccessful completion status is dropped and not delivered to its destination. The request that corresponds to the unsuccessful completion is retried by sending a new request for undeliverable data.

3.1.7.8 Error Reporting Changes

The PCIe Rev. 1.1 specification defines two changes to advanced error reporting. A (new) *Role Based Error Reporting* bit in the Device Capabilities register is set to 1b to indicate that these changes are supported by the 82599.

1. Setting the *SERR# Enable* bit in the PCI Command register also enables UR reporting (in the same manner that the *SERR# Enable* bit enables reporting of correctable and uncorrectable errors). In other words, the *SERR# Enable* bit overrides the *Unsupported Request Error Reporting Enable* bit in the PCIe Device Control register.
2. Changes in the response to some uncorrectable non-fatal errors detected in non-posted requests to the 82599. These are called Advisory Non-Fatal Error cases. For each of the errors listed, the following behavior is defined:
   - 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* corresponding 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 unserviced error.
   - An ERR_COR Message is sent if the *Correctable Error Reporting Enable* bit is set in the Device Control register. An ERROR_NONFATAL message is not sent for this error.

The following uncorrectable non-fatal errors are considered as advisory non-fatal errors:

- A completion with an Unsupported Request or Completer Abort (UR/CA) status that signals an uncorrectable error for a non-posted request. If the severity of the UR/CA error is non-fatal, the completer must handle this case as an advisory non-fatal error.
- 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 or to signal the error without attempting recovery. 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 is 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.
- Reception of a poisoned TLP. See Section 3.1.7.3.
- 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.

3.1.8 Performance Monitoring

The 82599 incorporates PCIe performance monitoring counters to provide common capabilities to evaluate performance. The 82599 implements four 32-bit counters to correlate between concurrent measurements of events as well as the sample delay and interval timers. The four 32-bit counters can also operate in a two 64-bit mode to count long intervals or payloads. Software can reset, stop, or start the counters (all at the same time).

Some counters operate with a threshold — the counter increments only when the monitored event crossed a configurable threshold (such as the number of available credits is below a threshold)

Counters operate in one of the following modes:

- Count mode — the counter increments when the respective event occurred
- Leaky bucket mode — the counter increments only when the rate of events exceeded a certain value. See Section 3.1.8.1 for more details.

The list of events supported by the 82599 and the counters *Control* bits are described in Section 8.2.3.3.

3.1.8.1 Leaky Bucket Mode

Each of the counters can be configured independently to operate in a leaky bucket mode. When in leaky bucket mode, the following functionality is provided:

- One of four 16-bit Leaky Bucket Counters (LBC) is enabled via the *LBC Enable* [3:0] bits in the PCIe Statistic Control register #1.
- The LBC is controlled by the GIO_COUNT_START, GIO_COUNT_STOP, GIO_COUNT_RESET bits in the PCIe Statistic Control register #1.

- The LBC increments every time the respective event occurs.
- The LBC is decremented every T μs as defined in the *LBC Timer* field in the PCIe Statistic Control registers.
- When an event occurs and the value of the LBC meets or exceeds the threshold defined in the *LBC Threshold* field in the PCIe Statistic Control registers, the respective statistics counter increments, and the LBC counter is cleared to zero.

3.2 SMBus

SMBus is a management interface for pass through and/or configuration traffic between an external Management Controller (MC) and the 82599.

3.2.1 Channel Behavior

The SMBus specification defines the maximum frequency of the SMBus as 100 KHz. However, the SMBus interface can be activated up to 400 KHz without violating any hold and setup time.

SMBus connection speed bits define the SMBus mode. Also, SMBus frequency support can be defined only from the EEPROM.

3.2.2 SMBus Addressing

The number of SMBus addresses that the 82599 responds to depends on the LAN mode (teaming/non-teaming). If the LAN is in teaming mode (fail-over mode), the 82599 is presented over the SMBus as one device and has one SMBus address. If the LAN is in non-teaming mode, the SMBus is presented as two SMBus devices on the SMBus (two SMBus addresses). In dual-address mode, all pass through functionality is duplicated on the SMBus address, where each SMBus address is connected to a different LAN port.

Note: Designers are not allowed to configure both ports to the same address. When a LAN function is disabled, the corresponding SMBus address is not presented to the MC.

The SMBus address method is defined through the *SMB Addressing Mode* bit in the EEPROM. The SMBus addresses are set using the *SMBus 0 Slave Address* and *SMBus 1 Slave Address* fields in the EEPROM.

Note: If single-address mode is selected, only the *SMBus 0 Slave Address* field is valid.

The SMBus addresses (those that are enabled from the EEPROM) can be re-assigned using the SMBus ARP protocol.

Besides the SMBus address values, all the previous parameters of the SMBus (SMBus channel selection, addressing mode, and address enable) can be set only through the EEPROM.

All SMBus addresses should be in Network Byte Order (NBO) with the most significant byte first.

3.2.3 SMBus Notification Methods

The 82599 supports three methods of signaling the external MC that it has information that needs to be read by the external MC:

- SMBus alert — Refer to Section 3.2.3.1.
- Asynchronous notify — Refer to Section 3.2.3.2.
- Direct receive — Refer to section Section 3.2.3.3.

The notification method that is used by the 82599 can be configured from the SMBus using the Receive Enable command. The default method is set from the *Notification Method* field in the LRXEN1 word from the EEPROM.

The following events cause the 82599 to send a notification event to the external MC:

- Receiving a LAN packet designated for the MC.
- Receiving a Request Status command from the MC that initiates a status response.
- The 82599 is configured to notify the external MC upon status changes (by setting the EN_STA bit in the Receive Enable command) along with one of the following events:
  - TCO Command Aborted
  - Link Status changed
  - Power state change
  - LinkSec indication

There can be cases where the external MC is hung and cannot respond to the SMBus notification. The 82599 has a timeout value defined in the EEPROM (refer to Section 6.4.4.3) to avoid hanging while waiting for the notification response. If the MC does not respond until the timeout expires, the notification is de-asserted.

3.2.3.1 SMBus Alert and Alert Response Method

The SMBus Alert# signal is an additional SMBus signal, which acts as an asynchronous interrupt signal to an external SMBus master. The 82599 asserts this signal each time it has a message that it needs the external MC to read and if the chosen notification method is the SMBus alert method.

Note: SMBus Alert# is an open-drain signal, which means that other devices beside the 82599 can be connected to the same alert pin and the external MC requires a mechanism to distinguish between the alert sources as follows:

The external MC responds to the alert by issuing an ARA cycle to detect the alert source device. The 82599 responds to the ARA cycle (if it was the SMBus alert source) and de-asserts the alert when the ARA cycle completes. Following the ARA cycle, the MC issues a Read command to retrieve the the 82599 message.

Note: Some MCs do not implement the ARA cycle transaction. These MCs respond to an alert by issuing a Read command to the 82599 (0xC0/0xD0 or 0xDE). The 82599 always responds to a Read command even if it is not the source of the notification. The default response is a status transaction. If the 82599 is the source of the SMBus alert, it replies to the read transaction.

The ARA cycle is an SMBus receive byte transaction to SMBus Address 0x18.

Note: The ARA transaction does not support PEC.

The alert response address transaction format is as follows:

1	7	1	1	8	1	1
S	Alert Response Address	Rd	A	Slave Device Address	A	P
	0001 100	0	0		1	

Figure 3-7 SMBus ARA Cycle Format

3.2.3.2 Asynchronous Notify Method

When configured using the asynchronous notify method, the 82599 acts as an SMBus master and notifies the external MC by issuing a modified form of the write word transaction. The asynchronous notify transaction SMBus address and data payload are configured using the Receive Enable command or by using the EEPROM defaults (see Section 6.4.3.19).

Note: The asynchronous notify is not protected by a PEC byte.

1	7	1	1	7	1	1	
S	Target Address	Wr	A	Sending Device Address		A	•••
	MC Slave Address	0	0	Manageability Slave SMBus Address	0	0	

8	1	8	1	1
Data Byte Low	A	Data Byte High	A	P
Interface	0	Alert Value	0	

Figure 3-8 Asynchronous Notify Command Format

3.2.3.3 Direct Receive Method

If configured, the 82599 has the capability to send the message it needs to transfer to the external MC, as a master over the SMBus instead of alerting the MC and waiting for it to read the message.

The message format is shown Figure 3-9. Note that the command that should be used is the same command that should be used by the MC in the Block Read command and the opcode that the 82599 puts in the data is the same as it would have put in the Block Read command of the same functionality. The rules for the *F* an *L* flags are also the same as in the Block Read command.

1	7	1	1	1	1	6	1	
S	Target Address	Wr	A	F	L	Command	A	• • •
	MC Slave Address	0	0	First Flag	Last Flag	Receive TCO Command 01 0000b	0	

8	1	8	1		1	8	1	1
Byte Count	A	Data Byte 1	A	• • •	A	Data Byte N	A	P
N	0		0		0		0	

Figure 3-9 Direct Receive Transaction Format

3.2.4 Receive TCO Flow

The 82599 is used as a channel for receiving packets from the network link and passing them to the external MC. The MC can configure the 82599 to pass specific packets to the MC (see Section 10.2). Once a full packet is received from the link and identified as a manageability packet that should be transferred to the MC, the 82599 starts the receive TCO transaction flow to the MC.

The maximum SMBus fragment length is defined in the EEPROM (see Section 6.4.4.2). The 82599 uses the SMBus notification method to notify the MC that it has data to deliver. The packet is divided into fragments, where the 82599 uses the maximum fragment size allowed in each fragment. The last fragment of the packet transfer is always the status of the packet. As a result, the packet is transferred in at least two fragments. The data of the packet is transferred in the receive TCO LAN packet transaction.

When SMBus alert is selected as MC notification method, the 82599 notifies the MC on each fragment of a multi-fragment packet.

When asynchronous notify is selected as the MC notification method, the 82599 notifies the MC only on the first fragment of a received packet. It is the MC's responsibility to read the full packet including all the fragments.

Any timeout on the SMBus notification results in discarding of the entire packet. Any NACK by the MC on one of the 82599's receive bytes also causes the packet to be silently discarded.

Since SMBus throughput is lower than the network link throughput, the 82599 uses an 8 KB internal buffer per LAN port, which stores incoming packets prior to being sent over the SMBus interface. The 82599 services back-to-back management packets as long as the buffer does not overflow.

The maximum size of the received packet is limited by the 82599 hardware to 1536 bytes. Packets larger then 1536 bytes are silently discarded. Any packet smaller than 1536 bytes is processed by the 82599.

Note: When the *RCV_EN* bit is cleared, all receive TCO functionality is disabled including packets directed to the MC as well as auto ARP processing.

3.2.5 Transmit TCO Flow

The 82599 is used as a channel for transmitting packets from the external MC to the network link. The network packet is transferred from the external MC over the SMBus, and then, when fully received by the 82599, is transmitted over the network link.

In dual-address mode, each SMBus address is connected to a different LAN port. When a packet received in SMBus transactions using the *SMBus 0 Slave Address*, it is transmitted to the network using LAN port 0 and is transmitted through LAN port 1 if received on *SMBus 1 Slave Address*. In single-address mode, the transmitted port is chosen according to the fail-over algorithm (see Section 10.2.2.2).

The 82599 supports packets up to an Ethernet packet length of 1536 bytes. SMBus transactions can be up to 240 bytes in length, which means that packets can be transferred over the SMBus in more than one fragment. In each command byte there are the *F* and *L* bits. When the *F* bit is set, it means that this is the first fragment of the packet and *L* means that it is the last fragment of the packet (when both are set, it means that the entire packet is in one fragment). The packet is sent over the network link only after all its fragments have been received correctly over the SMBus.

The 82599 calculates the L2 CRC on the transmitted packet, and adds its four bytes at the end of the packet. Any other packet field (such as XSUM) must be calculated and inserted by the external MC controller (the 82599 does not change any field in the transmitted packet, besides adding padding and CRC bytes). If the packet sent by the MC is bigger than 1536 bytes, then the packet is silently discard by the 82599.

The minimum packet length defined by the 802.3 specification is 64 bytes. The 82599 pads packets that are less than 64 bytes to meet the specification requirements (no need for the MC to do it). There is one exception, that is if the packet sent over the SMBus is less than 32 bytes, the MC must pad it for at least 32 bytes. The passing bytes value should be zero. Packets which are smaller then 32 bytes (including padding) are silently discarded by the 82599.

If the network link is down when the 82599 has received the last fragment of the packet, it silently discards the packet.

Note: Any link down event while the packet is being transferred over the SMBus does not stop the operation, since the 82599 waits for the last fragment to end to see whether the network link is up again.

The transmit SMBus transaction is described in Section 10.5.2.1.

3.2.5.1 Transmit Errors in Sequence Handling

Once a packet is transferred over the SMBus from the MC to the 82599 the *F* and *L* flags should follow specific rules. The *F* flag defines that this is the first fragment of the packet, and the *L* flag defines that the transaction contains the last fragment of the packet.

Table 3-9 lists the different option of the flags in transmit packet transactions.

Table 3-9 SMBus Transmit Sequencing

Previous	Current	Action/Notes
Last	First	Allowed – accept both.
Last	Not First	Error for current transaction. Current transaction is discarded and an abort status is asserted.
Not Last	First	Error for previous transaction. The previous transaction (until previous First) is discarded. The current packet is processed. No abort status is asserted.
Not Last	Not First	The 82599 can process the current transaction.

Note: Since every other Block Write command in TCO protocol has both F and L flags on, they cause flushing any pending transmit fragments that were previously received. When running the TCO transmit flow, no other Block Write transactions are allowed in between the fragments.

3.2.5.2 TCO Command Aborted Flow

Bit 6 in first byte of the status returned from the 82599 to the external MC indicates that there was a problem with previous SMBus transactions or with the completion of the operation requested in previous transaction.

The abort can be asserted due to any of the following reasons:

- Any error in the SMBus protocol (NACK, SMBus time outs).
- Any error in compatibility due to required protocols to specific functionality (RX Enable command with byte count not 1/14 as defined in the command specification).
- If the 82599 does not have space to store the transmit packet from the MC (in an internal buffer) before sending it to the link. In this case, all transactions are completed but the packet is discarded and the MC is notified through the *Abort* bit.
- Error in *F/L* bit sequence during multi-fragment transactions.
- The *Abort* bit is asserted after an internal reset to the 82599 manageability unit.

Note: The abort in the status does not always imply that the last transaction of the sequence was incorrect. There is a time delay between the time the status is read from the 82599 and the time the transaction has occurred.

3.2.6 Concurrent SMBus Transactions

Concurrent SMBus write transactions are not permitted. Once a transaction is started, it must be completed before additional transaction can be initiated.

3.2.7 SMBus ARP Functionality

The 82599 supports the SMBus ARP protocol as defined in the SMBus 2.0 specification. Note that the 82599 is a persistent slave address device each time its SMBus address is valid after power-up and loaded from the EEPROM. The 82599 supports all SMBus ARP commands defined in the SMBus specification, both general and directed.

Note: SMBus ARP can be disabled through EEPROM configuration (See Section 6.4.4.3).

3.2.7.1 SMBus ARP in Dual-/Single-Mode

The 82599 can operate in either single SMBus address mode or in dual SMBus address mode. These modes reflect on its SMBus-ARP behavior.

When working in single-address mode, the 82599 presents itself on the SMBus as one device, and responds to SMBus-ARP as only one device. In this case its SMBus address is SMBus address 0 as defined in EEPROM SMBus ARP addresses word (see Section 6.4.4.4). The device has only one Address Resolved (AR) and one Address Valid (AV) flag each. The vendor ID that is the Ethernet MAC address of the LAN's port, is taken from port 0 address.

In dual-address mode, the 82599 responds as two SMBus devices, meaning it has two sets of AR/AV flags (one for each port). The 82599 should respond twice to the SMBus-ARP master, one time for each port. Both SMBus addresses are taken from the SMBus ARP addresses word of the EEPROM. The Unique Device Identifier (UDID) is different between the two ports in the version ID field, which represent the Ethernet MAC address, which is different between the two ports. It is recommended for the 82599 to first answer as port 0, and only when the address is assigned, to answer as port 1 to the Get UDID command.

3.2.7.2 SMBus ARP Flow

SMBus-ARP flow is based on the status of two *AV*s and *AR*s:

- Address Valid — This flag is set when the 82599 has a valid SMBus address.
- Address Resolved — This flag is set when the 82599 SMBus address is resolved: SMBus address was assigned by the SMBus-ARP process.

Note: These flags are internal the 82599 flags and not shown to external SMBus devices.

Since the 82599 is a Persistent SMBus Address (PSA) device, the *AV* flag is always set, while the *AR* flag is cleared after power-up until the SMBus-ARP process completes. Since *AV* is always set, it means that the 82599 always has a valid SMBus address. The entire SMBus ARP Flow is described in Figure 3-10.

When the SMBus master needs to start the SMBus-ARP process, it resets (in terms of ARP functionality) all the devices on the SMBus, by issuing either Prepare to ARP or Reset Device commands. When the 82599 accepts one of these commands, it clears its *AR* flag (if set from previous SMBus-ARP process), but not its *AV* flag (The current SMBus address remains valid until the end of the SMBus ARP process).

A cleared *AR* flag means that the 82599 answers the following SMBus ARP transactions that are issued by the master. The SMBus master then issues a Get UDID command (General or Directed), to identify the devices on the SMBus. The 82599 responds to the Directed command all the time, and to the General command only if its *AR* flag is not set. After the Get UDID, the master assigns the 82599 SMBus address, by issuing Assign Address command. The 82599 checks whether the UDID matches its own UDID, and if there is a match it switches its SMBus address to the address assigned by the command (byte 17). After accepting the Assign Address command, the *AR* flag is set, and from this point (as long as the *AR* flag is set), the 82599 does not respond to the Get UDID General command, while all other commands should be processed even if the *AR* flag is set.

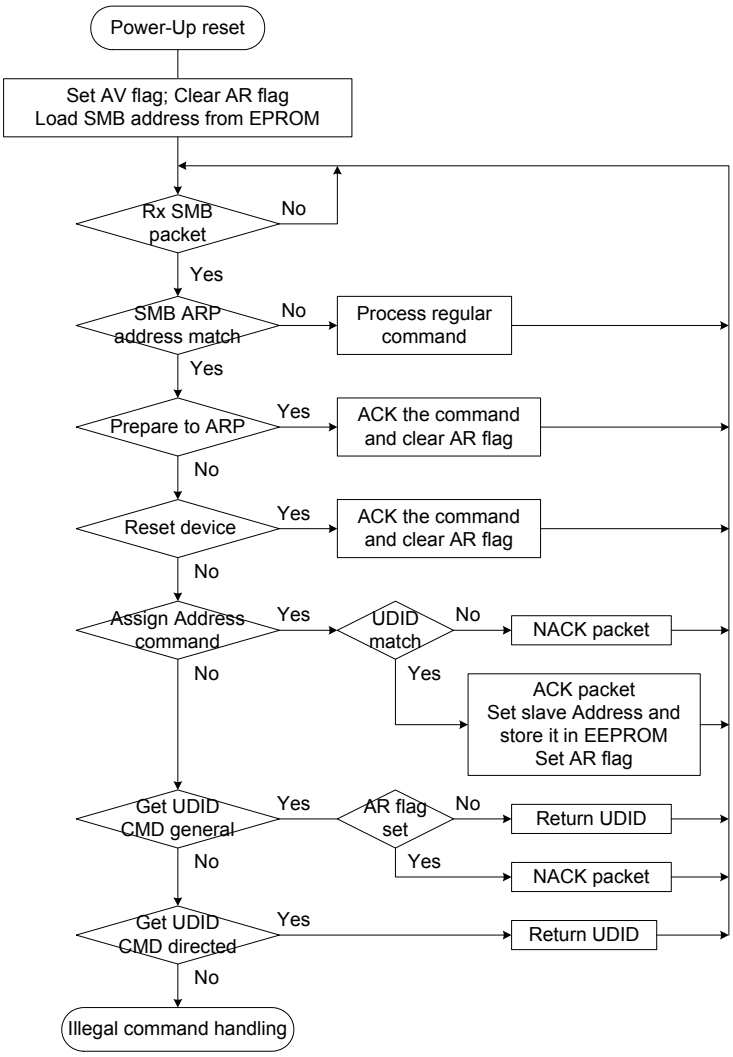


Figure 3-10 SMBus ARP Flow

3.2.7.2.1 SMBus ARP UDID Content

The UDID provides a mechanism to isolate each device for the purpose of address assignment. Each device has a unique identifier. The 128-bit number is comprised of the following fields:

1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes	4 Bytes
Device Capabilities	Version/ Revision	Vendor ID	Device ID	Interface	Subsystem Vendor ID	Subsystem Device ID	Vendor Specific ID
See as follows	See as follows	0x8086 As reflected in the Device ID field in the PCI config space	As reflected in the Device ID field in the PCI config space	0x0004	0x0000	0x0000	See as follows
MSB							LSB

Where:

- Interface: Identifies the protocol layer interfaces supported over the SMBus connection by the device.
  In this case, SMBus Version 2.0
  Constant value: 0x0004.
- Subsystem Fields: These fields are not supported and return zeros.

Device Capabilities: Dynamic and Persistent Address, *PEC Support* bit

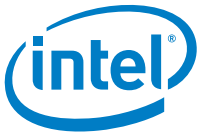
7	6	5	4	3	2	1	0
Address Type		Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	PEC Supported
0b	1b	0b	0b	0b	0b	0b	0b
MSB							LSB

Version/Revision: UDID Version 1, Silicon Revision

7	6	5	4	3	2	1	0
Reserved (0)	Reserved (0)	UDID Version			Silicon Revision ID		
0b	0b	001b			See as follows		
MSB							LSB

Silicon Revision ID:

Silicon Version	Revision ID
A0	000b
B0	001b



Vendor Specific ID:

Four LSB bytes of the device Ethernet MAC address. The device Ethernet MAC address is taken from the EEPROM LAN Core 0/1 Modules in the EEPROM (see Section 6.3.7.2). Note that in the 82599 there are two Ethernet MAC Addresses (one for each port).

1 Byte	1 Byte	1 Byte	1 Byte
Ethernet MAC Address, byte 3	Ethernet MAC Address, byte 2	Ethernet MAC Address, byte 1	Ethernet MAC Address, byte 0
MSB			LSB

3.2.8 LAN Fail-Over Through SMBus

In fail-over mode, the 82599 determines which ports are used for transmit/reactive (according to the configuration). LAN fail-over is tied to the SMBus addressing mode. When the SMBus is dual-address mode, the 82599 does not activate its fail-over mechanism (it ignores the fail-over register), and operates in two individual LAN ports. When the SMBus is in single-address mode, in PT mode, the 82599 operates in fail-over mode as described in section Section 10.2.2.2.

3.3 Network Controller — Sideband Interface (NC-SI)

In the 82599, the NC-SI interface is connected to an external MC. Note that the 82599 NC-SI interface meets the NC-SI specification as a PHY-side device.

3.3.1 Electrical Characteristics

The 82599 complies with the electrical characteristics defined in the NC-SI specification.

The 82599 NC-SI behavior is configured by the 82599 on power-up:

- The output driver strength for the NC-SI output signals is configured by the EEPROM *RMM Out Buffer Strength* field (default = 0x1F).
- The NC-SI topology is loaded from the EEPROM (point-to-point or multi-drop with the default being point-to-point).

The 82599 dynamically drives its NC-SI output signals as required by the sideband protocol:

- On power up, the 82599 floats the NC-SI outputs.
- If the 82599 operates in point-to-point mode, then the 82599 starts driving the NC-SI outputs at some time following power up.
- If the 82599 operates in a multi-drop mode, the 82599 drives the NC-SI outputs as configured by the MC.

3.3.2 NC-SI Transactions

Compatible with the NC-SI specification.

3.4 EEPROM

3.4.1 General Overview

The 82599 uses an EEPROM device for storing product configuration information. The EEPROM is divided into three general regions:

Hardware accessed — loaded by the 82599 hardware after power-up, PCI reset de-assertion, D3 to D0 transition, or software reset. Different hardware sections in the EEPROM are loaded at different events. See further details on power-up and reset sequences in Section 4.0.

Firmware Area — Includes structures used by the firmware for management configuration in its different modes.

Software accessed — used by software only. The meaning of these registers as listed here is a convention for software only and is ignored by the 82599.

3.4.2 EEPROM Device

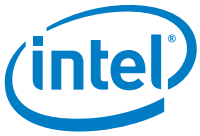
The EEPROM interface supports an SPI interface and. It expects the EEPROM to be capable of 5 MHz operation.

The 82599 is compatible with many sizes of 4-wire serial EEPROM devices. All EEPROM's are accessed in 16-bit data words only. For EEPROM size information, see Section 11.6.2.

The 82599 automatically determines the address size to be used with the SPI EEPROM it is connected to, and sets the EEPROM *Size* field of the EEPROM/FLASH Control and Data register (EEC.EE_ADDR_SIZE) field appropriately. Software can use this size to determine how to access the EEPROM. The exact size of the EEPROM is determined within one of the EEPROM words.

3.4.3 EEPROM Vital Content

The EEPROM contains several main types of vital content: pre-boot, pre-operating system and pre-driver parameters, manageability related structures, hardware default parameters, and driver default parameters. The 82599 must have the EEPROM to auto-load these settings.



3.4.4 Software Accesses

The 82599 provides two different methods for software access to the EEPROM.

- Use the built-in controller to read the EEPROM
- Access the EEPROM directly using the EEPROM's 4-wire interface

In addition, the Vital Product Data (VPD) area of the EEPROM can be accessed via the VPD capability structure of the PCIe.

Software can use the EEPROM Read (EERD) register to cause the 82599 to read a word from the EEPROM that the software can then use. To do this, software writes the address to read to the *Read Address* (EERD.ADDR) field and then simultaneously writes a 1b to the *Start Read* bit (EERD.START). The 82599 reads the word from the EEPROM, sets the *Read Done* bit (EERD.DONE), and puts the data in the *Read Data* field (EERD.DATA). Software can then poll the EEPROM Read register until it sees the *Read Done* bit set, then use the data from the *Read Data* field.

Note: Any words read this way are not written to the 82599's internal registers.

Software can also directly access the EEPROM's 4-wire interface through the EEPROM/Flash Control (EEC) register. It can use this for reads, writes, or other EEPROM operations.

To directly access the EEPROM, software should follow these steps:

1. Write a 1b to the *EEPROM Request* bit (EEC.EE_REQ).
2. Read the *EEPROM Grant* bit (EEC.EE_GNT) until it becomes 1b. It remains 0b as long as hardware is accessing the EEPROM.
3. Write or read the EEPROM using the direct access to the 4-wire interface as defined in the EEPROM/Flash Control and Data (EEC) register. The exact protocol used depends on the EEPROM placed on the board and can be found in the appropriate EEPROM datasheet.
4. Write a 0b to the *EEPROM Request* bit (EEC.EE_REQ).

Note: Each time the EEPROM is not valid (blank EEPROM or wrong signature), software should use the direct access to the EEPROM through the EEC register.

3.4.5 Signature Field

The only way the 82599 has to tell if an EEPROM is present is by trying to read the EEPROM. The 82599 first reads the EEPROM Control word at word address 0x000000 and at address 0x000800. It then checks the signature value at bits 7 and 6 in both addresses. If bit 7 is 0b and bit 6 is 1b in one of the two addresses, it considers the EEPROM to be present and valid. It then reads the additional EEPROM words and programs its internal registers based on the values read. Otherwise, it ignores the values it reads from that location and does not read any other words.

3.4.6 Protected EEPROM Space

The 82599 provides a mechanism for a hidden area in the EEPROM of the host. The hidden area cannot be read or write accessed via the EEPROM registers in the CSR space. It can be accessed only by the manageability subsystem.

After the EEPROM was configured to be protected, changing bits that are protected require specific manageability instructions with an authentication mechanism. This mechanism is defined in the firmware documentation.

3.4.6.1 Initial EEPROM Programming

In most applications, initial EEPROM programming is done directly on the EEPROM pins. Nevertheless, it is desired to enable existing software utilities (accessing the EEPROM via the host interface) to initially program the entire EEPROM without breaking the protection mechanism. Following a power up sequence, the 82599 reads the hardware initialization words in the EEPROM. If the signature in both word addresses 0x000000 and 0x000800 is not equal to 01b the EEPROM is assumed as non-programmed. There are two effects of a non-valid signature:

1. The 82599 does not read any further EEPROM data and sets the relevant registers to default.
2. The 82599 enables host write (and read) access to any location in the EEPROM via the EEPROM CSR registers.

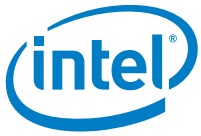
3.4.6.2 EEPROM Protected Areas

The 82599 defines two protected areas in the EEPROM. The first area is words 0x00-0x0F. These words hold the basic configuration and the pointers to all other configuration sections. The second area is a programmable size area located at the end of the EEPROM and assigned with protecting the appropriate sections that should be blocked for changes.

3.4.6.3 Activating the Protection Mechanism

Following a device initialization, the 82599 reads the Init control 1 word from the EEPROM sectors 0 and 1. The 82599 turns on the protection mechanism if this word contains a valid signature (equals to 01b) and bit 4 (EEPROM protection) is set to 1b. Once the protection mechanism is turned on, words 0x00-0x0F area become write-protected and the hidden area that is defined by word 0x0 becomes read/write protected to host access.

Note: Although possible by configuration, it is prohibited that the software sections in the EEPROM is included as part of the EEPROM protected area.



3.4.6.4 Non Permitted Access to Protected Areas in the EEPROM

This section refers to EEPROM accesses by the host via the EEC (bit banging) or EERD (parallel read access) registers. Following a write access to the protected areas in the EEPROM (word 0x0 and the hidden area defined by word 0x0), hardware responds properly on the PCIe bus but does not initiate any access to the EEPROM. Following a read access to the hidden area in the EEPROM (as defined by word 0x0), hardware does not access the EEPROM and returns meaningless data to the host.

Notes: Using the bit banging access, the SPI EEPROM can be accessed in a burst mode by providing opcode, address, and then read or write data for multiple bytes. Hardware inhibits any attempt to access the protected EEPROM locations even in burst accesses.

Software should not access the EEPROM in a burst write mode starting in a non-protected area and continue to a protected one, or vice versa. In such a case, it is not guaranteed that the write access to the non-protected area takes place.

3.4.7 EEPROM Recovery

The EEPROM contains fields that if programmed incorrectly might affect the functionality of the 82599. The impact might range from an incorrect setting of some function (like LED programming), via disabling of entire features (such as no manageability) and link disconnection, to inability to access the device via the regular PCIe interface.

The 82599 implements a mechanism that enables recovery from a faulty EEPROM no matter what the impact is, using an SMBus message that instructs the firmware to invalidate the EEPROM.

This mechanism uses an SMBus message that the firmware is able to receive in all modes, no matter what is the content of the EEPROM. After receiving this message, firmware clears word 0x0 including the signature. Afterwards, the BIOS/operating system initiates a reset to force an EEPROM auto-load process that fails and enables access to the device.

Firmware is programmed to receive such a command only from PCIe reset until one of the functions changes its status from D0u to D0a. Once one of the functions moves to D0a it can be safely assumed that the device is accessible to the host and there is no further need for this function. This reduces the possibility of malicious software using this command as a back door and limits the time the firmware must be active in non-manageability mode.

The command is sent on a fixed SMBus address of 0xC8. The format of the SMBus Block Write command is as follows:

Function	Command	Data Byte
Release EEPROM	0xC7	0xB6

Notes: This solution requires a controllable SMBus connection to the 82599.

In case more than one the 82599 is in a state to accept this solution, all of the the 82599 devices connected to the same SMBus accept the command. The devices in D0u state release the EEPROM.

After receiving a release EEPROM command, firmware should keep its current state. It is the responsibility of the programmer updating the EEPROM to send a firmware reset if required after the full EEPROM update process is done.

An additional command is introduced to enable the EEPROM write directly from the SMBus interface to enable the EEPROM modification (writing from the SMBus to any MAC CSR register). The same rules as for the Release EEPROM command that determine when firmware accepts this command apply to this command as well.

The command is sent on a fixed SMBus address of 0xC8. The format of the SMBus Block Write command is as follows:

Function	Command	Byte Count	Data 1	Data 2	Data 3	Data 4	...	Data 7
EEPROM Write	0xC8	7	Config Address 2	Config Address 1	Config Address 0	Config Data MSB	...	Config Data LSB

The most significant bit in Configuration address 2 indicates which port is the target of the access (0 or 1). The 82599 always enables the manageability block after power up. The manageability clock is stopped only if the manageability function is disabled in the EEPROM and one of the functions had transitioned to D0a; otherwise, the manageability block gets the clock and is able to wait for the new command.

This command enables writing to any MAC CSR register as part of the EEPROM recovery process. This command can be used to write to the EEPROM and update different sections in it.

3.4.8 EEPROM Deadlock Avoidance

The EEPROM is a shared resource between the following clients:

1. Hardware auto read.
2. LAN port 0 and LAN port 1 software accesses.
3. Manageability-firmware accesses.

When accessing the EEPROM, software and manageability-firmware should use the EEPROM parallel access. On this interface, hardware schedules the actual accesses to the EEPROM, avoiding starvation of any client. The bit banging interface does not guarantee fairness between the clients, therefore it should be avoided in nominal operation as much as possible. When write accesses to the EEPROM are required the software or manageability should access the EEPROM one word at a time releasing the interface after each word.

3.4.9 VPD Support

The EEPROM image can contain an area for VPD. This area is managed by the OEM vendor and does not influence the behavior of the hardware. Word 0x2F of the EEPROM image contains a pointer to the VPD area in the EEPROM. A value of 0xFFFF means VPD is not supported and the VPD capability does not appear in the configuration space.

The maximum area size is 256 bytes but can be smaller. The VPD block is built of a list of resources. A resource can be either large or small. The structure of these resources are listed in the following tables.

Table 3-10 Small Resource Structure

Offset	0	1 — n
Content	Tag = 0xxx,xyyyb (Type = Small(0), Item Name = xxxx, length = yy bytes)	Data

Table 3-11 Large Resource Structure

Offset	0	1 - 2	3 — n
Content	Tag = 1xxx,xxxxb (Type = Large(1), Item Name = xxxxxxxx)	Length	Data

The 82599 parses the VPD structure during the auto-load process following PCIe reset in order to detect the read only and read/write area boundaries. The 82599 assumes the following VPD fields with the limitations listed in Table 3-12.

Table 3-12 VPD Structure

Tag	Length (Bytes)	Data	Resource Description
0x82	Length of identifier string	Identifier	Identifier string.
0x90	Length of RO area	RO data	VPD-R list containing one or more VPD keywords.
0x91	Length of RW area	RW data	VPD-W list containing one or more VPD keywords. This part is optional.
0x78	N/A	N/A	End tag.

VPD structure limitations:

- The structure must start with a tag equal to 0x82. If the 82599 does not detect a value of 0x82 in the first byte of the VPD area or the structure does not follow the information listed inTable 3-12, it assumes the area is not programmed and the entire 256 bytes area is read only.
- The RO area and R/W area are both optional and can appear in any order. A single area is supported per tag type. See PCI 3.0 specification Appendix I for details of the different tags.
- If a VPD-W tag is found, the area defined by its size is writable via the VPD structure.

- Both read and write sections on the VPD area must be Dword aligned (for example, each tag must start on Dword boundaries, and each data field must end on Dword boundary). Write accesses to Dwords that are only partially in the R/W area are ignored. VPD software is responsible to make the right alignment to enable a write to the entire area.
- The structure must end with a tag equal to 0x78. The tag must be word aligned.
- The VPD area is accessible for read and write via the regular EEPROM mechanisms pending the EEPROM protection capabilities enabled. The VPD area can be accessed through the PCIe configuration space VPD capability structure listed in Table 3-12. Write accesses to a read only area or any accesses outside of the VPD area via this structure are ignored.

3.5 Flash

The 82599 provides an interface to an external serial Flash/ROM memory device. This Flash/ROM device can be mapped into memory space for each LAN device through the use of Base Address Registers (BARs). EEPROM bits associated with each LAN device selectively disable/enable whether the Flash can be mapped for each LAN device by controlling the BAR register advertisement and write ability.

3.5.1 Flash Interface Operation

The 82599 provides two different methods for software access to the Flash.

Using the legacy Flash transactions the Flash is read from, or written to, each time the host CPU performs a read or a write operation to a memory location that is within the Flash address mapping or upon boot via accesses in the space indicated by the Expansion ROM Base Address register. All accesses to the Flash require the appropriate command sequence for the device used. Refer to the specific Flash data sheet for more details on reading from or writing to Flash. Accesses to the Flash are based on a direct decode of CPU accesses to a memory window defined in either:

- Memory CSR + Flash Base Address register (PCIe Control register at offset 0x10).
- The Expansion ROM Base Address register (PCIe Control register at offset 0x30).

The 82599 controls accesses to the Flash when it decodes a valid access.

Note: Flash read accesses must always be assembled by the 82599 each time the access is greater than a byte-wide access.

The 82599 byte reads or writes to the Flash take about 2 µs time. The device continues to issue retry accesses during this time.

The 82599 supports only byte writes to the Flash.

Another way for software to access the Flash is directly using the Flash's 4-wire interface through the Flash Access (FLA) register. It can use this for reads, writes, or other Flash operations (accessing the Flash status register, erase, etc).