7.13 Fibre Channel over Ethernet (FCoE)

7.13.1 Introduction

Fibre Channel (FC) is the predominant protocol used in Storage Area Networks (SAN). Fibre Channel over Ethernet (FCoE) is used to connect an Ethernet storage initiator and legacy FC storage targets.

The FC protocol is based on high reliability of the communication link between the initiator and the storage target. It assumes an extremely low error rate of 10^{-12} and no packet drop. DCB extends Ethernet through class-based flow control in such a way that FC-like no-drop is guaranteed as required by FC. Doing so, FC protocol can be transposed to an Ethernet link by Layer 2 encapsulation that is defined by the FCoE protocol. Figure 7-46 shows a connection between an FCoE initiator and legacy FC targets.

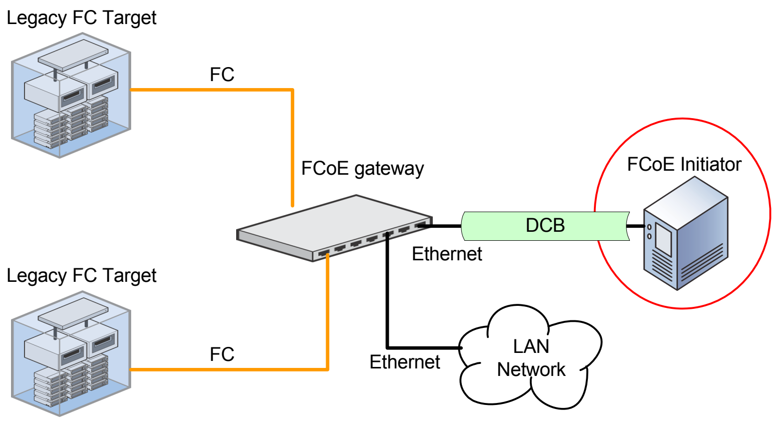


Figure 7-46 Connecting an FCoE Initiator to FC Targets

Existing FC HBAs used to connect between an FC initiator and FC targets provide full offload of the FC protocol to the initiator to maximize storage performance. In order to compete with this market, the 82599 offloads the main data path of I/O Read and Write commands to the storage target.

7.13.1.1 FC Terminology

Useful background on FC framing and its Ethernet encapsulation can be found in Section A.5. More comprehensive material can be found in the FIBRE CHANNEL FRAMING AND SIGNALING-2 (FC-FS-2) specification. Following are some of the most common terms used extensively in the sections that describe the FCoE functionality.

FC Exchange - Complete FC read or FC write flow. It starts with a read or write request by the initiator (the host system) until it receives a completion indication from the target (the remote disk).

FC Sequence - An FC exchange is composed of multiple FC sequences. An FC sequence can be single or multiple frames that are sent by the initiator or the target. Also, each FC sequence has a unique sequence ID.

FC Frame - FC frames are the smallest units sent between the initiator and the target. The FC-FS-2 specification defines the maximum frame size as 2112 bytes. Each FC frame includes an FC header and optional FC payload. It also may include extended headers and FC optional headers. Extended headers other than Virtual Fabric Tagging (VFT) are not expected in an FCoE network and FC optional headers are not used in most cases as well.

Data Frame - FC frames that carry read or write data.

FCP_RSP Frame - FC control frames are sent from the target to the initiator, which defines the completion of an FC read or write exchange.

7.13.2 FCoE Transmit Operation

Transmit FCoE offload is enabled by setting the TUCMD.FCoE bit in the transmit context descriptor. The 82599 supports the following offload capabilities: FC CRC calculation and insertion, FC padding insertion and FC segmentation. These capabilities are described in the following sections.

7.13.2.1 FCoE Transmit Cross Functionality

After setting the TUCMD.FCoE bit, hardware digests the packet's content before it is sent to the wire. In this case, software must enable hardware offload for additional tasks as follows:

Table 7-87 FCoE Transmit Cross Functionality

Cross Function	Requirements
Ethernet CRC insertion	Software must enable Ethernet CRC insertion by setting the *IFCS* bit in the transmit data descriptor. The Ethernet CRC covers the entire packet. Enabling FCoE offloading, hardware modifies the packet content and must also adjust the Ethernet CRC.
LinkSec offload	LinkSec encapsulation covers the entire Ethernet packet payload (it includes both FCoE content and Ethernet padding). When packets carry LinkSec encapsulation on the wire, LinkSec offload by hardware should be activated.
VLAN header	It is assumed that any FCoE has a VLAN header. In the case of double VLAN mode, the packet must have the two VLAN headers.
SNAP packet	The 82599 does not provide FCoE offload for FCoE frame over SNAP.
Traffic rate control	FC traffic relies on a high quality link that guarantees no packet loss. It is expected that any lost traffic protocols supported by the network are enabled by the 82599 as well.
FC and PFC	
Virtualization	It is expected that the VMM abstract the FCoE functionality to the VM(s). FCoE setting and FCoE traffic is expected only by the VMM accessing the LAN via the PF.
TCP/IP and UDP/IP offload	FCoE traffic is L2 traffic (not over IP). Any setting of TCP/IP and UDP/IP offload capabilities are not applicable and do not impact FCoE offload functions.
Transmit descriptors	Software must use the advanced transmit descriptor to activate either FC CRC offload or TSO functionality.

7.13.2.2 FC Padding Insertion

FC frames always consist of a whole number of four bytes. If user data is not composed of a whole number of four bytes, then the FC frames contain padding bytes with a zero value. The length of the padding bytes can be any number between zero to three so together with the user data, the length of the FC frames has a whole number of four bytes. The length of the padding bytes is indicated by software in the *Fill Bytes* field in the FC header. This field is used by the receiving end node (target) to extract these bytes. Hardware does not use this field to identify the required length of the padding bytes. Instead, it checks the transmit buffer size indicated by the *PAYLEN* field in the transmit data descriptor. The length of the padding bytes added by hardware equals:

2's complement {two LS bits of (PAYLEN minus MACLEN)}. While PAYLEN is defined in the Tx data descriptor and MACLEN is defined in the Tx context descriptor.

The 82599 auto-pads the frame with the required zero bytes when FCoE offload is enabled (TUCMD.FCoE bit is set). In TSO, padding bytes are added only on the last frame since the MSS must be a whole number of four bytes.

7.13.2.3 SOF Placement

During a single send, the *SOF* field is taken as is from the FCoE header in the data buffer.

7.13.2.4 EOF Insertion

The 82599 automatically inserts the *End of Frame* field when the TUCMD.FCoE bit in the transmit context descriptor is set. The EOF codes that are inserted into the transmitted packets are stored in the TEOFF register. The TEOFF register contains four EOF codes named EOF0...EOF3 that are supported by the transmit FCoE offload. By default, these values are programmed into the following values: EOF0 = EOFn; EOF1 = EOFt; EOF2 = EOFni; EOF3 = EOFa. The *EOF* flag in the *FCoEF* field in the transmit context descriptor define an index value as listed in the Table 7-88.

Table 7-88 EOF Codes in Single Send

EOF Bits in the Context Descriptor (ORIE bit in the Context Descriptor must be set to 1b)	00	01	10	11
Inserted EOF Code	EOF0 (EOFn)	EOF1 (EOFt)	EOF2 (EOFni)	EOF3 (EOFa)

7.13.2.5 FC CRC Insertion

FC CRC calculation is one of the most CPU intensive tasks in large transactions. The 82599 offloads the FC CRC calculation when the *FCoE* bit is set in the *TUCMD* field within the transmit context descriptor. The 82599 calculates and adds the FC CRC before packet transmission but after the required FC padding bytes are already added.

The CRC polynomial used by the FC protocol is the same one as used in FDDI and Ethernet as shown in the following equation. While CRC bytes are transmitted in big endian byte ordering (MS byte first on the wire):

$$X_{32}+X_{26}+X_{23}+X_{22}+X_{16}+X_{12}+X_{11}+X_{10}+X_8+X_7+X_5+X_4+X_2+X+1.$$

The size of FCoE payload on which FC CRC is calculated is indicated in the context and data descriptors as follows. Figure 7-47 specifies the FCoE frame and the relevant parameters to CRC calculation.

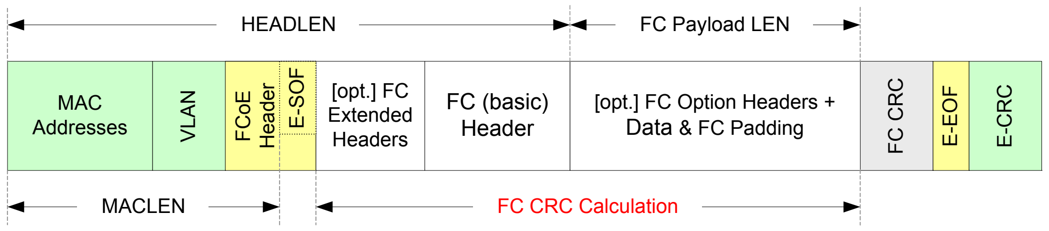


Figure 7-47 FCoE Frame and Relevant Transmit Descriptor Parameters

FC CRC Calculation Beginning

FC CRC calculation starts after the FCoE header. It equals to byte offset of MACLEN + 4, while the *MACLEN* field in the transmit context descriptor is the byte offset of the last Dword in the FCoE header that contains the SOF flag.

FC CRC Calculation End

FC CRC calculation ends at the end of the FC Payload LEN shown in Figure 7-47 (eight bytes before the Ethernet CRC).

7.13.2.6 Host Data Buffers Content for a Single Packet Send

The Table 7-89 lists the data prepared by software when transmit FCoE offload is enabled (the *FCoE* bit in the *TUCMD* field is set in the transmit context descriptor).

Table 7-89 Transmit FCoE Packet Data Provided by Software (for TUCMD.FCoE = 1)

Ethernet MAC Addresses	VLAN Header	FCoE Header	FC Frame (provided by software)		
			FC Header	FC Option Header(s)	Opt. Data

Listed below are fields in the transmitted FCoE frame that are not included in the data buffers (in host memory) as shown in Figure 7-89.

VLAN Header — The VLAN header could be part of the data buffer or in the transmit descriptor depending on *VLE* bit in the *CMD* field in the transmit descriptor.

EOF — The EOF is defined by the *EOF* fields and *ORIE* bit in the context descriptor (more details in Section 7.2.3.2.3)

FC-CRC — The 82599 calculates and inserts the FC CRC bytes.

FC-Padding — The 82599 calculates the padding length and inserts these bytes as required (all zeros).

Ethernet CRC — Insertion should be enabled by the *IFCS* bit in transmit data descriptor.

LinkSec Header and Digest — When the link is secured by LinkSec, then LinkSec offload must be enabled and the LinkSec encapsulation is added by hardware.

7.13.2.7 FCoE Transmit Segmentation Offload (TSO)

FCoE segmentation enables the FCoE software to initiate a transmission of multiple FCoE packets up to a complete FC sequence with a single header in host memory (single instruction). It is activated by using the advanced Tx context descriptor (DTYP equals 0010b) and setting both the TUCMD.FCoE in the context descriptor and setting the DCMD.TSE bit in the transmit data descriptor. The 82599 splits the transmitted content to multiple packets as defined by the *MSS* field in the Tx context descriptor.

TSO Parameters

- The frame header includes the Ethernet MAC addresses, VLAN Tag, FCoE header and the FC header. The header size is defined in the context descriptor by the HEADLEN and MACLEN as illustrated.
- The *SOF* and *EOF* fields are defined by the SOF, ORIS, EOF and ORIE fields in the context descriptor as described in Section 7.13.2.3 and Section 7.13.2.4.
- MSS – the maximum segment size in the context descriptor that define the FC data (payload) size on each packet other than the last frame which can be smaller.

7.13.2.7.1 Host Data Buffers Content for TSO Offload

The Figure 7-48 shows the data in host memory when FCoE TSO is activated. The TSO header is repeated on all frames of the TSO. The header includes static and dynamic fields that are modified by hardware from packet-to-packet. The payload size is reflected in all frames.

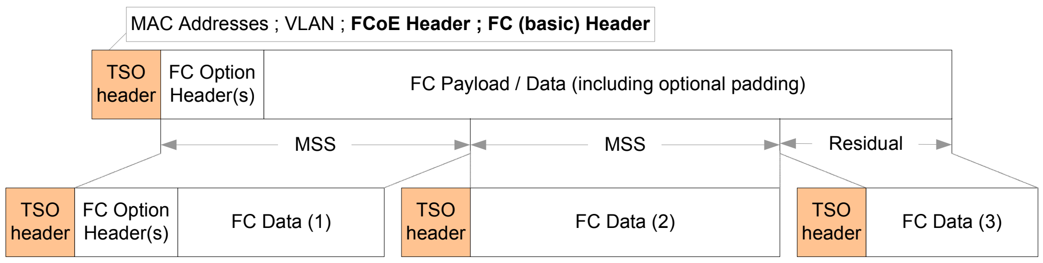


Figure 7-48 FCoE TSO Provided by the FCoE Driver

FCoE Header

The FCoE packet header must not span more than two buffers. For best bus use it is recommended that the header be located in a single buffer (the first one).

- Ethernet MAC addresses are the source and destination Ethernet MAC addresses

- VLAN tag can be provided by the driver as part of the packet header or as part of the data descriptor.
- FCoE header (shown in Figure 7-48) includes the FCoE Ethernet type, FCoE Version and SOF flag. Software should leave the *SOF* fields as zero while hardware inserts it according to the *SOF* and *ORIS* bits in the Tx context descriptor.
- FC (basic) header.

FCoE TSO — Payload

- FC option headers.
- FC data to be segmented
- The payload may or may not include the optional FC padding bytes. Hardware adds any required padding bytes not included in the data buffers according to the PAYLEN field in the data descriptor.

Modified fields between consecutive frames within TSO are described in the following sections.

7.13.2.7.2 Dynamic Start of Frame in TSO

During TSO the *SOF* field in the data buffer is replaced by hardware according to the values of the *SOF* and *ORIS* bits in the transmit context descriptor. In this case the value of the *SOF* field in the data buffer is ignored (for future expansion software should set it to zero). The SOF codes that are inserted to the transmitted packets are stored in the TSOFF register. The TSOFF register contains four SOF codes named as SOF0...SOF3 that are supported by the transmit FCoE offload. By default these values are programmed to the following values: SOF0 = SOFi2; SOF1 = SOFi3; SOF2 = SOFn2; SOF3 = SOFn3. The SOF flag and *Orientation Start* (ORIS) bit in the *FCoEF* field in the transmit context descriptor define an index value. This index is used to extract the SOF code that is inserted to the packet as listed in the Table 7-90. The *ORIS* bit defines if the TSO starts an FC sequence or if the first frame on the FC sequence is already sent.

Table 7-90 SOF Codes in TSO

SOF **Bit in the Context Descriptor**	***ORIS*** **Bit in the Context Descriptor**	**SOF Code in the First Frame**	**SOF Code in Other Frames**	**SOF Code while TSO = Single Frame**
1 (Class 3)	1 (sequence start)	SOF1 (SOFi3)	SOF3 (SOFn3)	SOF1 (SOFi3)
1 (Class 3)	0 (not a sequence start)	SOF3 (SOFn3)	SOF3 (SOFn3)	SOF3 (SOFn3)
0 (Class 2)	1 (sequence start)	SOF0 (SOFi2)	SOF2 (SOFn2)	SOF0 (SOFi2)
0 (Class 2)	0 (not a sequence)	SOF2 (SOFn2)	SOF2 (SOFn2)	SOF2 (SOFn2)

7.13.2.7.3 Dynamic FC Header fields in TSO

F_CTL Table 7-91 lists those fields in the F_CTL that are modified between consecutive frames of a TSO. If a TSO is transmitted by a single packet all F_CTL fields are taken from the data buffer (as if it is the last frame in the TSO).

Table 7-91 F_CTL Codes in TSO

F_CTL Bits	last frame in TSO when the *ORIE* bit in the Tx context descriptor is set.	Any other frame
Fill Bytes (1:0)	Taken from the F_CTL(1:0) in the data buffer. It defines the length of the FC padding required to make the FC data a complete multiply of four bytes.	00b
Continue Sequence Condition (7:6)	Taken from the F_CTL(7:6) in the data buffer. The continue sequence condition is meaningful only if F_CTL(19) is set and F_CTL(16) is cleared.	00b
Sequence Initiative (16)	Taken from the F_CTL(16) in the data buffer. The sequence initiative is meaningful only if F_CTL(19) is also set.	0b
End Sequence (19)	Taken from the F_CTL(19) in the data buffer. The end sequence should be set to 1b by software only if the frame is the last one of a sequence.	0b

DF_CTL	Table 7-92 lists those fields in the DF_CTL that can be modified between consecutive frames of a TSO. Note that the ESP Header presence bit is not listed in this table. When ESP Hea*der* is present, software must not use a TSO that spans across multiple packets. If a TSO is transmitted by a single packet all DF_CTL fields are taken from the data buffer (as if it is the first frame in the TSO).

Table 7-92 DF_CTL Codes in TSO

DF_CTL Fields	1st frame in TSO when *ORIS* bit in the Tx context descriptor is set.	Any other frame
Device Header Indication (1:0)	Taken from the data buffer.	00b
Association Header Indication (4)	Taken from the data buffer.	0b
Network Header Indication (5)	Taken from the data buffer.	0b

SEQ_CNT	SEQ_CNT in the first frame is taken from the SEQ_CNT field in the FC header in the data buffers. On any other frame, the value of SEQ_CNT is incremented by one from its value in the previous frame. The SEQ_CNT wrap-to-zero after reaching a value of 65,535.
PARAM	The PARAM field in the first frame is taken from the *PARAM* field in the FC header in the data buffers. If the FCoEF.PARINC bit is set in the transmit context descriptor, the value of the PARAM becomes dynamic. In that case, the *PARAM* is incremented by hardware by the MSS value on each frame. Software should set the FCoEF.PARINC bit when the *PARAM* field indicates the data offset (*Relative Offset Present* bit in the F_CTL field is set).

7.13.2.7.4 Dynamic End Of Frame Fields

FC_CRC	Calculated and inserted on each frame as described in section Section 7.13.2.5.
FC_Padding	Calculated the number of required padding bytes and inserted them on the last frame as described in section Section 7.13.2.2.
EOF	As explained for a single send, the EOF flag is appended to the transmitted packets while values are taken from the TEOFF register. The FCoE flag index to the TEOFF register is defined by the *EOF* flag and *Orientation End (ORIE*) bit in the *FCoEF* field in the transmit context descriptor as listed in Table 7-93.

Table 7-93 EOF Codes in TSO

EOF Bits in the Context Descriptor	ORIE Bit in the Context Descriptor	Last Frame of the TSO or TSO in Single Frame	Other Frames of the TSO
00 (EOFn)	0 (not a sequence end)	EOF0 (EOFn)	EOF0 (EOFn)
00 (EOFn)	1 (sequence end)	EOF1 (EOFt)	EOF0 (EOFn)
Else = Reserved	N/A	N/A	N/A

7.13.3 FCoE Receive Operation

The 82599 can offload the following tasks from the CPU while processing FCoE receive traffic: FC CRC check, receive coalescing and Direct Data placement (DDP). These offload options are described in the sections that follow.

DDP functionality is not provided for control packets or data packets that do not meet DDP criteria (described later in the sections that follow). In those cases, hardware posts the packets to the legacy Rx queues as is (header and trailer are not stripped including SOF, EOF, FC padding and FC CRC bytes). When DDP functionality is enabled, only the FC payload is posted to the user buffers. If the packet's header should be indicated to the legacy Rx queues, all bytes starting at the destination Ethernet MAC address until the FC header and optionally FC header(s) inclusive are posted to the legacy buffer.

7.13.3.1 FCoE Receive Cross Functionality

FCoE receive offload capabilities coexist with other functions in the 82599 are listed as follows:

Table 7-94 FCoE Receive Cross Functionality

Cross Function	Requirements
Ethernet CRC check	There is no enforcement on save bad frames policy. In the case of save bad frames, packets with bad Ethernet CRC are posted to the legacy receive queue even if DDP is enabled. FC payload of bad packets are never posted directly to the user buffers.
Ethernet padding extraction	There is no enforcement on the Ethernet padding extraction. When DDP is enabled, hardware posts the FC payload to the user buffers. When DDP is not enabled the entire packets are posted to the legacy receive queues with or without the Ethernet padding according to the device setting.
LinkSec offload	LinkSec encapsulation covers the entire Ethernet packet payload. If the traffic includes LinkSec, hardware must process first the LinkSec encapsulation uncovering the FCoE plain text to the FCoE offload logic. If the LinkSec processing is not enabled and the packets include LinkSec encapsulation, then the packets are posted to the matched legacy receive queue. If the LinkSec processing is enabled but fails for any reason, the packet can still be posted to the matched legacy queue according to save bad frames policy.
VLAN header	It is assumed that any FCoE has a VLAN header. In the case of double VLAN mode, the packet must have the two VLAN headers.
SNAP packet	The 82599 does not provide FCoE offload for FCoE frame over SNAP.
FC and PFC	FC traffic relies on a high-quality link that guarantees no packet loss. It is expected that any lost traffic protocols supported by the network is enabled by the 82599 as well.
Virtualization	It is expected that VM(s) generate FC write requests to the VMM. FCoE setting and FCoE traffic is expected only by the VMM accessing the physical function.
TCP/IP and UDP/IP offload	FCoE traffic is L2 traffic (not over IP). Any setting of TCP/IP and UDP/IP offload capabilities are not applicable and do not impact FCoE offload functions.
Jumbo frames	Maximum expected clear text FC frame size is 2140 bytes (FC header + FC payload + FC CRC). Adding optional FC crypto, plus FCoE encapsulation, plus optional LinkSec encapsulation packet might exceed the 2200 bytes. In order to enable FCoE traffic, jumbo packet reception should be enabled.
Receive descriptors in the legacy Rx queues	When FC CRC offload or DDP functionality are enabled, software must use the advanced descriptors in the associated legacy Rx queues (SRRCTL.DESCTYPE = 001b). The legacy Rx buffers must be larger than the maximum expected packet size so any Rx packets span on a single buffer.

7.13.3.2 FC Receive CRC Offload

FC CRC calculation is one of the most CPU intensive tasks in TSO transactions. The 82599 offloads the receive FC CRC integrity check while trashing the CRC bytes and FC padding bytes.

The 82599 recognizes FCoE frames in the receive data path by their FCoE Ethernet type and the FCoE version in the FCoE header. The Ethernet type that hardware associates with FCoE is defined in the ETQF register by setting the *FCoE* bit with a specific Ethernet type value. The supported FCoE versions by the Rx offload logic are defined by FCRXCTRL.FCOEVER. FCoE packets that do not match the previously described Ethernet type and FCoE versions are ignored by the Rx FCoE logic.

The 82599 reconstructs the FC CRC while processing the incoming bytes and compares it against the received FC CRC. The frame is considered a good FC packet if the previous comparison matches and it is considered as a bad FC packet otherwise.

The FC CRC integrity check is meaningful only if all the following conditions are met:

- The received frame contains a correct Ethernet CRC
- If the received frame includes LinkSec encapsulation then LinkSec offload must be enabled and LinkSec integrity is found OK.

The length of the FC padding bytes that hardware trashes are defined in the *Fill Bytes* field in the FC frame control (F_CTL). The *Fill Bytes* field can have any value between zero to three that makes the FC frame a whole number of Dwords. It is expected that the *Fill Bytes* field would be zero except for last data frames within a sequence.

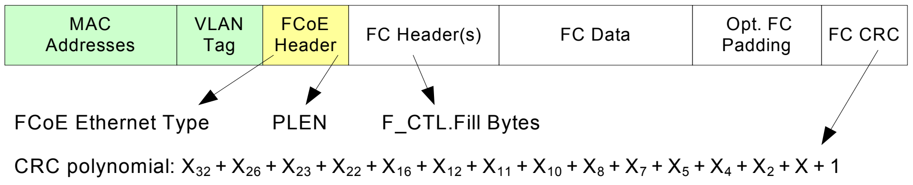

MAC Addresses
VLAN Tag
FCoE Header
FC Header(s)
FC Data
Opt. FC Padding
FC CRC
FCoE Ethernet Type
PLEN
F_CTL.Fill Bytes
CRC polynomial: $X_{32} + X_{26} + X_{23} + X_{22} + X_{16} + X_{12} + X_{11} + X_{10} + X_8 + X_7 + X_5 + X_4 + X_2 + X + 1$

Figure 7-49 Relevant FCoE and FC Fields for CRC Receive Offload

7.13.3.3 Large FC Receive

Large FC receive includes two types of offloads. The 82599 can save a data copy by posting the received FC payload directly to the kernel storage cache or the user application space (in the remainder of the document there is no difference between the two cases and it is named as user buffers). When the packet's payload are posted directly to the user buffers their headers can still be posted to the legacy receive queues. The 82599 saves CPU cycles by reducing the data copy and also minimize CPU processing by posting only the packet's headers that are required for software.

Figure 7-50 shows the mapping of received FCoE frames to the legacy Rx queue and the user buffers. Figure 7-51 shows a top level overview of the large FC receive flow. The remaining sections detail the large FC receive functionality as follows:

- Enabling large FC receive — Section 7.13.3.3.1
- FC read exchange flow — Section 7.13.3.3.2
- FC write exchange flow — Section 7.13.3.3.3
- FCoE receive filtering (Frame types and rules) — Section 7.13.3.3.5, Section 7.13.3.3.10 and Section 7.13.3.3.12
- User descriptors — Section 7.13.3.3.7
- Header posting to the legacy receive queues and FC exceptions — Section 7.13.3.3.13 and Section 7.13.3.3.14
- Interrupts — Section 7.13.3.3.15

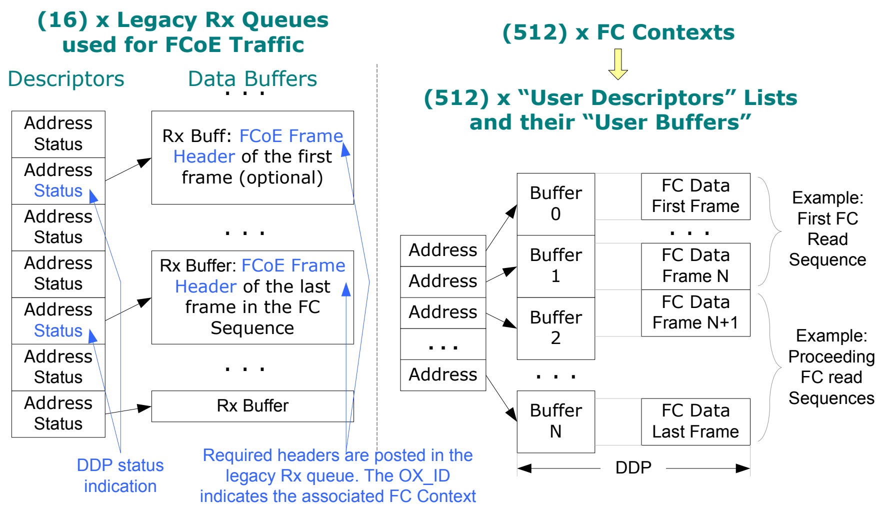
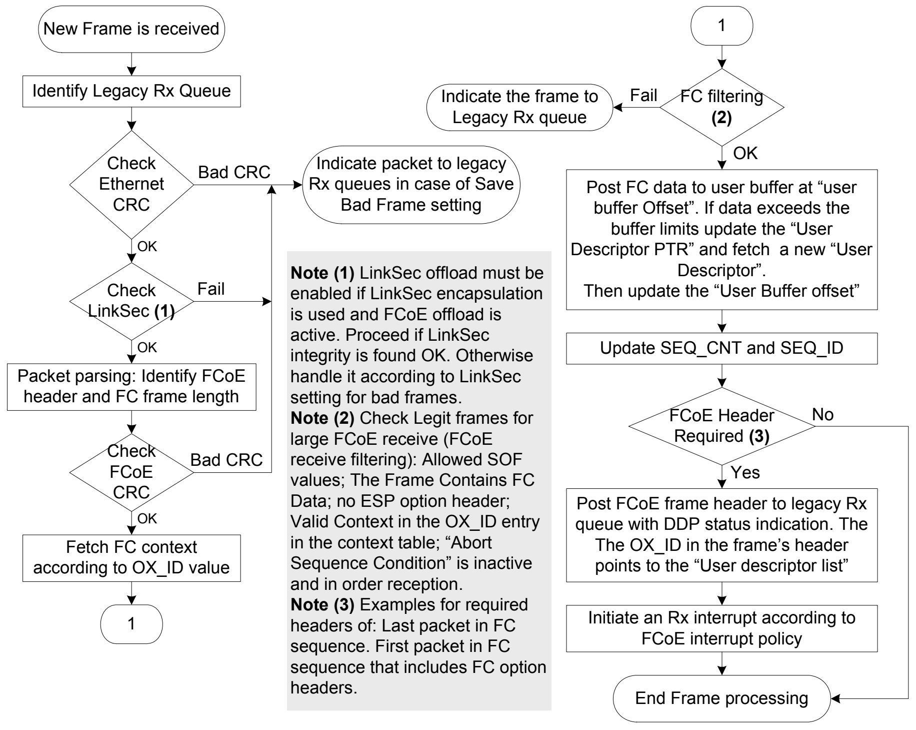

(16) x Legacy Rx Queues used for FCoE Traffic
Descriptors
Data Buffers
Address Status
Rx Buff: FCoE Frame Header of the first frame (optional)
Rx Buffer: FCoE Frame Header of the last frame in the FC Sequence
Rx Buffer
DDP status indication
Required headers are posted in the legacy Rx queue. The OX_ID indicates the associated FC Context
(512) x FC Contexts
(512) x "User Descriptors" Lists and their "User Buffers"
Address
Buffer 0
Buffer 1
Buffer 2
Buffer N
FC Data First Frame
FC Data Frame N
FC Data Frame N+1
FC Data Last Frame
Example: First FC Read Sequence
Example: Proceeding FC read Sequences
DDP

Figure 7-50 Large FC Reception to User Buffers and Legacy Rx Queue


New Frame is received
Identify Legacy Rx Queue
Check Ethernet CRC
Bad CRC
OK
Indicate packet to legacy Rx queues in case of Save Bad Frame setting
Check LinkSec (1)
Fail
Packet parsing: Identify FCoE header and FC frame length
Check FCoE CRC
Fetch FC context according to OX_ID value
1
FC filtering (2)
Indicate the frame to Legacy Rx queue
Post FC data to user buffer at "user buffer Offset". If data exceeds the buffer limits update the "User Descriptor PTR" and fetch a new "User Descriptor". Then update the "User Buffer offset"
Update SEQ_CNT and SEQ_ID
FCoE Header Required (3)
No
Yes
Post FCoE frame header to legacy Rx queue with DDP status indication. The OX_ID in the frame's header points to the "User descriptor list"
Initiate an Rx interrupt according to FCoE interrupt policy
End Frame processing

Note (1) LinkSec offload must be enabled if LinkSec encapsulation is used and FCoE offload is active. Proceed if LinkSec integrity is found OK. Otherwise handle it according to LinkSec setting for bad frames.
Note (2) Check Legit frames for large FCoE receive (FCoE receive filtering): Allowed SOF values; The Frame Contains FC Data; no ESP option header; Valid Context in the OX_ID entry in the context table; "Abort Sequence Condition" is inactive and in order reception.
Note (3) Examples for required headers of: Last packet in FC sequence. First packet in FC sequence that includes FC option headers.

Figure 7-51 FCoE Large Receive Flow Diagram

7.13.3.3.1 Enabling Large FC Receive

Large FC receive offload is enabled per each outstanding read or write exchange by programming the FCoE context table with the flow parameters. Setting the FC context for read or write exchange is done at run time. It is expected that a read context is programmed before the read request is initiated to the remote target and write context is programmed before the target sends the ready indication to the initiator. Unless the FC context is invalidated, software must not modify it in the middle of a transaction (see Section 7.13.3.3.5 for details on context invalidation). For more details on FCoE initialization flow see Section 4.6.9.

7.13.3.3.2 FC Read Exchange Flow

Figure 7-52 shows an example of an FC (class 3) read request. This flow is detailed in this section.

1. The software checks if the read request can use large FC receive offload depending on FC context resources and some criteria as listed in Section 7.13.3.3.5. Section 7.13.3.3.12 describes a proposed software flow to manage the FC contexts.
2. If the previous conditions are not met, software can initiate the FC read request according the flow described in Figure 7-52 while the received frames are posted to the legacy receive queues.

- If the previous conditions are met, software locks the relevant user buffers (the target buffers for the FC read request) and program the FC context table. It then initiates the FC read request according the flow shown in Figure 7-52. The payload of the received frames is posted directly to the user buffers. Some of the packet's headers (only the required ones) are posted to the legacy receive queues. The FC header in the packet's header contains the OX_ID field. This field indicates to software its context and its user buffer list. During nominal operation, all packets' headers except packets with FC optional headers are trashed by the hardware minimizing software overhead.

3. The target sends the FCP_RSP frame type indicating the completion of the read exchange. As a response, the hardware invalidates the FC read context (if it was used) and indicates the number of bytes posted directly to the user buffers in the receive descriptor (see Section 7.13.3.3.13). Software indicates the read completion to the application.

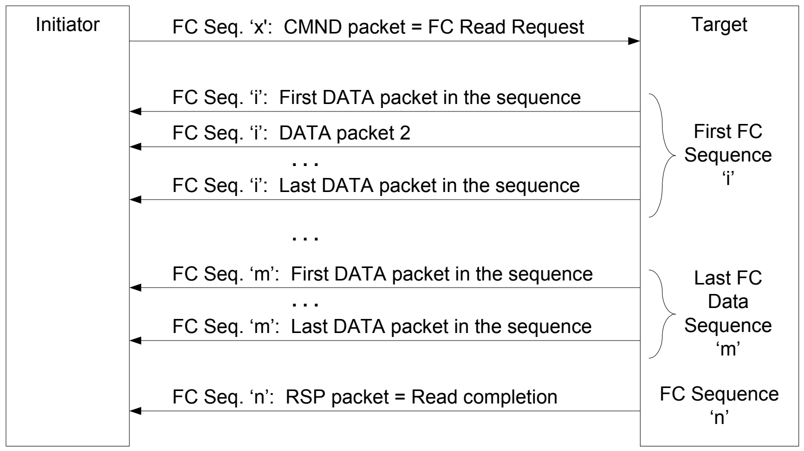

Initiator
Target
FC Seq. 'x': CMND packet = FC Read Request
FC Seq. 'i': First DATA packet in the sequence
FC Seq. 'i': DATA packet 2
. . .
FC Seq. 'i': Last DATA packet in the sequence
First FC Sequence 'i'
. . .
FC Seq. 'm': First DATA packet in the sequence
. . .
FC Seq. 'm': Last DATA packet in the sequence
Last FC Data Sequence 'm'
FC Seq. 'n': RSP packet = Read completion
FC Sequence 'n'

Figure 7-52 Example for FC Class 3 Read Exchange Flow

7.13.3.3.3 FC Write Exchange Flow

Figure 7-53 shows an example of an FC (class 3) write request (on which the Seq_CNT starts from zero on each new sequence). This flow is detailed in the sections that follow.

1. The host (originator) sends an FC write request to the target (responder).
2. Software in the target checks if the write request can use large FC receive offload depending on FC context resources and some criteria as listed in Section 7.13.3.3.5.
3. If the previous conditions are met, software can use DDP for this FC write exchange.
4. The target software locks the relevant user buffers (the target buffers for the FC write request) and program the FC context table. It then initiates the FC ready indication to the host.
5. As a response, the host sends the data frames to be written to the target. The frames are received in the target. If DDP is used, the FC payload is posted directly to the user buffers while "most" (see additional details below) packet's headers are trashed minimizing software overhead.
6. The host marks the last data frame it was requested to send by setting the *Sequence Initiative* bit in the F_CTL field.
7. The target identifies the last data frame and invalidates the DDP context. As indicated above, during nominal operation, "most" packet's headers are trashed. Only headers that have meaningful content are posted to host memory as: Headers of packets with FC optional headers and the header of the last packet in a sequence with active sequence initiative bit are posted to the legacy receive queues. The hardware indicates the number of bytes posted directly to the user buffers in the receive descriptor (see Section 7.13.3.3.13). Note that the FC header contains the RX_ID field that can be used by software to identifies its associated DDP context and user buffer list.
8. The target may repeat step 4, which is followed by step 5 until the entire requested data is transferred.

9. The target sends the FCP_RSP frame indicating to the initiator the completion of the write exchange.

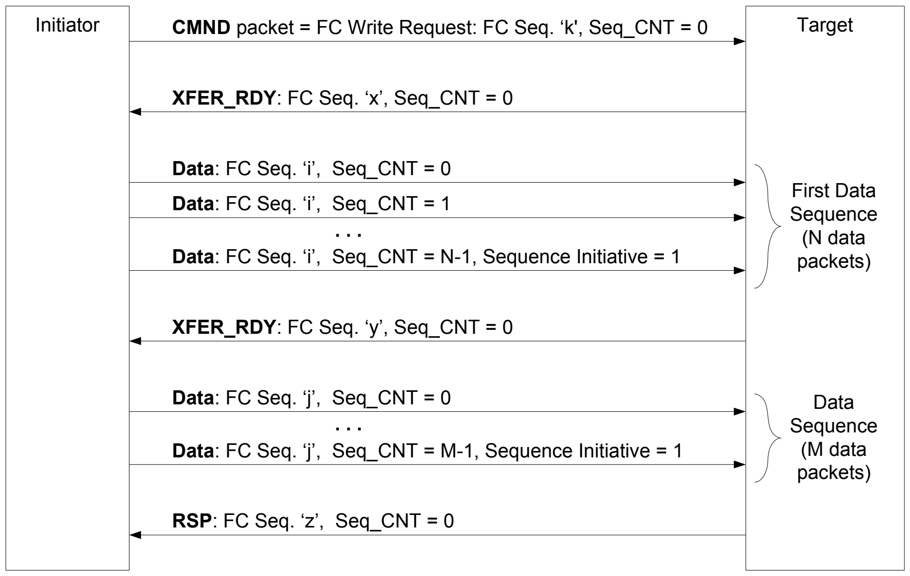

Initiator
Target
CMND packet = FC Write Request: FC Seq. 'k', Seq_CNT = 0
XFER_RDY: FC Seq. 'x', Seq_CNT = 0
Data: FC Seq. 'i', Seq_CNT = 0
Data: FC Seq. 'i', Seq_CNT = 1
. . .
Data: FC Seq. 'i', Seq_CNT = N-1, Sequence Initiative = 1
First Data Sequence (N data packets)
XFER_RDY: FC Seq. 'y', Seq_CNT = 0
Data: FC Seq. 'j', Seq_CNT = 0
. . .
Data: FC Seq. 'j', Seq_CNT = M-1, Sequence Initiative = 1
Data Sequence (M data packets)
RSP: FC Seq. 'z', Seq_CNT = 0

Figure 7-53 Example for FC Class 3 Write Exchange Flow

7.13.3.3.4 EOF and SOF Flags identification

As part of the DDP functionality, hardware identifies the SOF and EOF flags in the received packets. The flags identification is based on a setting of the RSOFF and REOFF registers. These registers are identical to the TSOFF and TEOFF registers and should be programmed by software to the same values.

7.13.3.3.5 FCoE Receive Filtering

Received FCoE frames are associated to one of the legacy receive queues according to the scheme described in Section 7.1.2. When the legacy receive queue is enabled, large FC receive functionality is enabled as well if a matched FC receive context is defined. The data is posted to the user buffers that are pointed to by the FC receive context. Some of the headers of these frames that are required for the data processing are posted to the legacy receive queue (see Section 7.13.3.3.13).

FCoE frames that carry FC class 3 or class 2 data can be posted to large receive buffers if they meet the following conditions:

- If the received packet carries LinkSec encapsulation it must be offloaded (and de-capsulated) by hardware.
- The FC context table contains valid context that matches the exchange ID in the received frame. Hardware checks the RX_ID for write data packets sent by the initiator. These packets are identified by the *Exchange Context* bit in the F_CTL header equals zero (originator of exchange). Hardware checks the OX_ID for read response data packets sent by the target. These packets are identified by the *Exchange Context* bit in the F_CTL header equals one (responder of exchange).

- Frames are identified as FCoE frame type according to the Ethernet type in the FCoE header. The Ethernet type that hardware associates with FCoE is defined in the ETQF registers by setting the *FCoE* bit with a specific Ethernet type value.
- The FC frame carry class 2 or class 3 content as defined by the SOF flag. The SOF in the FCoE header equals SOFi2 or SOFn2 or SOFi3 or SOFn3.
- The FCoE version in the received frame is equal or lower than FCRXCTRL.FCOEVER.
- The frame contains data content (with data payload) as defined in the *Routing Control* field (R_CTL) in the FC header:
  - R_CTL.Information (least significant four bits) equals 0x1 (solicited data)
  - R_CTL.Routing (most significant four bits) equals 0x0 (device data)
  - Frames that do not contain device data are not posted to the user buffers. Still these frames are compared against the expected SEQ_ID and SEQ_CNT in the FC context and update these parameters as described in Section 7.13.3.3.5.
- The FC frame does not include ESP header (bit 6 in the DF_CTL field within the FC header is cleared). Frames that include ESP option headers are posted to the legacy receive queue. For good use of hardware resources, software should not program the large FC receive context table with flows that carry an ESP header.
- The FC frame does not include any FC extended headers. For good use of hardware resources, software should not program the large FC receive context table with flows that carry extended headers.
- The first packet received to a new context is identified as the first FC frame in the exchange. This packet is expected to have the SOFi2 or SOFi3 codes. The SEQ_ID on the first packet may have any value.
- The frame is received in order as defined in Section 7.13.3.3.7 and does not carry any exception errors as defined in Section 7.13.3.3.14.
- The first frame on each FC sequence is identified by the SOFi2 or SOFi3 codes in the *SOF* field in the FCoE header. It is expected that the SEQ_ID is changed for any new sequence.
- The last frame on each FC sequence is identified by an active *End Sequence* flag in the F_CTL field in the FC header. It is expected to receive the EOFt code in the *EOF* field; however, hardware does not check this rule.

Other frames (that do not meet the previous conditions) are posted to the legacy receive queues according to the generic Rx filtering rules.

7.13.3.3.6 DDP Context

Hardware can provide DDP offload for up to 512 concurrent outstanding FC read or write exchanges. Each exchange has an associated FC context in hardware. Contexts are identified by the exchange ID (OX_ID for FC read and RX_ID for FC write). The exchange ID is a 16-bit field so that a system could theoretically generate up to 64 K concurrent outstanding FC read requests and 64 K concurrent outstanding FC write requests. Hardware contains 512 contexts for the 512 concurrent outstanding exchanges. Using exchange ID values between 0 to 511, software can benefit from the DDP offload. Any exchange ID value in the range of 0 to 511 can be used for either read or write exchange but not for both.

The FC context is a set of parameters used to identify a frame and its user buffers in host memory. The context parameters are split into two categories (according to the internal hardware implementation): DMA context (FCPTRL, FCPTRH, FCBUFF and FCDMARW registers) and filter context (FCFLT, FCPARAM and FCFLTRW register) as listed in Table 7-95 and shown in Figure 7-54.

Software should program both the DMA context and filter context making the context usable. During reception, hardware updates some of the parameters if the packet matches all criteria detailed in Section 7.13.3.3.5. Initialization values and the updated ones are listed in this section.

Table 7-95 Large FC Context Table

Exchange ID	DMA Context (FCPTRL, FCPTRH, FCBUFF, FCDMARW)		Filter Context (FCFLT and FCFLTRW)	
	DMA Flags	**User Descriptor**	**Filter Flags**	**In Order Reception**
0	Valid, First, Count	Size, Offset, Pointer	Valid, First	Seq_ID, Seq_CNT,PARAM
1	Valid, First, Count	Size, Offset, Pointer	Valid, First	Seq_ID, Seq_CNT,PARAM
2	Valid, First, Count	Size, Offset, Pointer	Valid, First	Seq_ID, Seq_CNT,PARAM
. . .	. . .		. . .	
511	Valid, First, Count	Size, Offset, Pointer	Valid, First	Seq_ID,Seq_CNT,PARAM

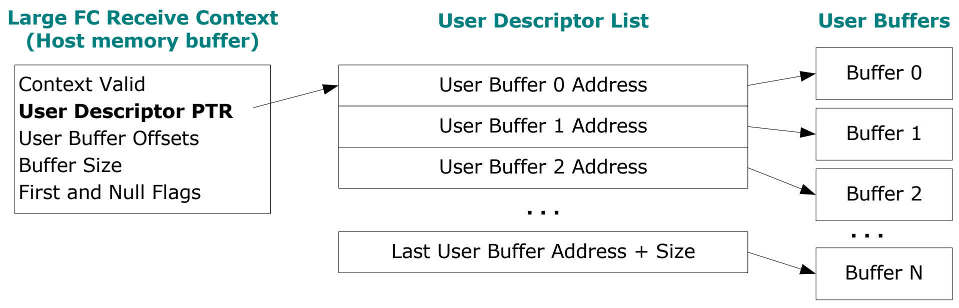


Figure 7-54 Large FC Receive Context Related to the User Buffers

DMA Context Valid (1 bit) and Filter Context Valid (1 bit) — These bits indicates the validity of this context.

Note: During programming time, software should enable first the DMA context. When software disables a context it should invalidate first the filter context. See more details on context invalidation in Section 7.13.3.3.10.

Filter First (1 bit) and DMA First (1 bit) — The first received frame that matches an active context in the filter unit is marked by the filter. This marking is used by the DMA unit as an indication that reception to this context has been started. The DMA context does not accept packets from the filter unit unless it received successfully the packet that was marked as the first one (see the section on exception handling in Section 7.13.3.3.14). The Filter First flag should be cleared by software when programming the context. Hardware sets this bit when the filter unit recognizes the first packet that matches a valid context. The DMA First bit should be cleared by software when programming the context.

Hardware sets this bit when the DMA unit received packet that matches a valid context and marked as first by the filter unit.

Buffer Count (8 bit) — This field defines the number of remaining user buffers in the list. At programming time, software sets the buffer count to the number of the allocated user buffers. During reception, hardware decrements the buffer count as each of them completes. The number of active buffers equals the buffer count value while 0x00 equals 256.

Buffer Size (2 bit) — This field defines the user buffer size used in this context. It can be 4 KB, 8 KB, 16 KB or 64 KB. All buffers except the first one and the last one are full size. The address of all buffers is aligned to the buffer size in the context. The first buffer may start at a non-zero offset. The size of the last buffer may be smaller than the buffer size as defined by the last buffer size parameter.

User Buffer Offset (16 bit) — This field defines the byte offset within the current buffer to which the next packet should be posted. At context programming, the software sets the user buffer offset to the beginning of the first buffer. During reception, hardware updates this field at the end of each packet processing for the next received packet.

Last User Buffer Size (16 bit) — This field defines the size of the last user buffer in byte units.

User Descriptor PTR (8 byte) — The user buffers are indicated by a list of pointers named as user descriptors (see Section 7.13.3.3.9 for a description of the user descriptors). The user descriptor PTR in the FC context is a pointer to the user descriptor list. At programming time, software sets the user descriptor PTR to the beginning of the user descriptor list. During reception, hardware increments the user descriptor PTR by eight (the size of the user descriptor) when it completes a buffer and requires the next one.

SEQ_ID (8bit) — The sequence ID identifies the sequence number sent by the target. An FC read or write exchange can be composed of multiple sequences depending on the target implementation. The SEQ_ID has a different value for each sequence and does not necessarily increment sequentially. Hardware uses the SEQ_ID for checking in-order reception as described in Section 7.13.3.3.7. Hardware updates the SEQ_ID in the context table according to the value of the SEQ_ID in the incoming frame. The initialization value during programming could be of any value. For future compatibility software should set it to zero.

SEQ_CNT (16 bit) — SEQ_CNT is an index of the expected FC frames within a sequence or within the entire exchange depending on the target implementation. Hardware uses the SEQ_CNT for checking in order reception as described in Section 7.13.3.3.7. On read context, software should initialize SEQ_CNT to zero. On write context, software should initialize SEQ_CNT to SEQ_CNT + 1 of the last packet of the same exchange received from the initiator. For each in-order reception, hardware sets SEQ_CNT in the context to the value of the received SEQ_CNT + 1.

PARAM (32 bit) — The PARAM field in the FC header may indicate the data offset within the FC IO exchange. It is indicated as an offset by the *Relative Offset Present* bit in the F_CTL field in the FC header. In this case, the *PARAM* field indicates the expected value of the next received packet. At programming time, software should initialize it to zero. During reception, hardware increments the *PARAM* by the size of the FC payload if it is used as an offset. The FC payload size equals the packet size minus the length of its header and trailer. While the header for this purpose includes all bytes starting at the Ethernet destination address up to and including the basic FC header, the trailer includes the FC CRC, FC padding, EOF including the three reserved bytes, and the Ethernet CRC.

7.13.3.3.7 In Order Reception Checking

Hardware checks in-order reception by *SEQ_ID*, *SEQ_CNT* and *PARAM* fields. These parameters should meet the expected values (as follows) in order to pass in-order reception's criteria.

PARAM — When the PARAM field is used as an offset (as indicated by the Relative Offset Present bit in the F_CTL field in the FC header), the PARAM field in the received packet should be the same as the PARAM field in the FC context. Software should initialize this parameter to the expected received value (equals to zero in read exchanges).

SEQ_ID, SEQ_CNT — SEQ_ID identifies the FC sequence and SEQ_CNT is the FC frame index within the entire exchange or within the sequence (according to specific vendor preference). SEQ_CNT in the received packet could be either the same as the SEQ_CNT in the FC context or it could start from zero for new SEQ_ID, which is different than the SEQ_ID in the context. Software should initialize SEQ_CNT to the expected received value (equals zero in read exchanges). SEQ_ID on the first packet is always assumed to be a new value even if by chance it equals to the initial value in the context.

7.13.3.3.8 Accessing the Large FC Receive Context

The 82599 supports a large number of FC contexts while each context contains about 16 bytes. In order to save consumed memory space, the FC context is accessed by indirect mapping. This section describes how the DMA and filter contexts are accessed. The DMA context is consist of the FCPTRL, FCPTRH and FCBUFF registers while read and write accesses are controlled by the FCDMARW register. The filter context is consist of the FCFLT register while read and write accesses are controlled by the FCFLTRW register.

DMA Context Programming — Software should program the FCPTRL, FCPTRH and FCBUFF registers by the required setting. It then programs the FCDMARW register with the following content:

- FCoESEL should be set by the required context index (OX_ID or RX_ID values)
- The *WE* bit is set to 1b for write access while the *RE* bit is set to 0b.
- LASTSIZE should be set to the relevant value for the context

DMA Context Read — Software should program the FCDMARW register as follows and then read the context on the FCPTRL, FCPTRH, FCBUFF and FCDMARW registers

- Software should initiate two consecutive write cycles to the FCDMARW register with the following setting: FCoESEL should be set to the required FCoE read index while both *WE* and *RE* should be set to 0b.
- FCoESEL should be set by the required context index (OX_ID or RX_ID values).
- *RE* bit should be set to 1b for read access while *WE*, and *LASTSIZE* fields are set to 0b.
- LASTSIZE should be set to 0b. It is ignored by hardware when the *RE* bit is set to 1b. When reading *FCDMARW* the *LASTSIZE* field reflects the context content.

Filter Context Programming — Software should program the FCFLT register by the required setting. It then programs the FCFLTRW register with the following content:

- FCoESEL should be set by the required context index (OX_ID or RX_ID values).
- WE bit is set to 1b for a write access while RE bit is set to 0b.

Filter Context Read — Software should program the FCFLTRW register as follows and then read the context on the FCFLT register:

- FCoESEL should be set by the required context index (OX_ID or RX_ID values).
- RE bit should be set to 1b for a read access while WE bit is set to 0b.

7.13.3.3.9 User Descriptor Structure and User Descriptor List

The buffers in host memory could be either user application memory or storage cache named as user buffers. In both cases the buffers must be locked (against software) and converted to physical memory up front.

The user descriptor list is a contiguous list of pointers to the user buffers. The buffers are aligned to their size as defined in the FC context. The first buffer can start at a non-zero offset as the software defines it in the FC context. All other buffers start at a zero offset. The last buffer can be smaller than the full size as defined in the FC context.

Table 7-96 FC User Descriptor

63	0
User buffer address defined in byte units. N LS bits must be set to zero while N equals 12 for a 4 KB buffer size, 13 for 8 KB buffer size, 14 for 16 KB buffer size and 16 for 64 KB buffer size.	

7.13.3.3.10 Invalidating FC Receive Context

During nominal activity, hardware invalidates autonomously the FC contexts. The target indicates a completion of an FC read by sending the FCP_RSP frame. Hardware identifies the FCP_RSP frame and invalidates the FC context that matches the OX_ID in the incoming frame. The FCP_RSP frame is posted to the legacy Rx queues with appropriate status indication. Hardware identifies the FCP_RSP frame by the following criteria:

- The frame is identified as FCoE frame by its Ethernet type
- R_CTL.Information (least significant four bits) equals 0x7 (command status)
- R_CTL.Routing (most significant four bits) equals 0x0 (device data)

Context that is invalidated autonomously by hardware is indicated by setting the *FCSTAT* field in the receive descriptor to 10b. When software gets this indication it can unlock the user buffers instantly and re-use the context for a new FC exchange.

In some erroneous cases software might invalidate a context before a read exchange completes (such as a time out event). In such cases, software should clear the *Filter Context Valid* bit and then the *DMA Context Valid* bit. Hardware invalidates the context at a packet's boundaries. Therefore, after software clears the *DMA Context Valid* bit, software should either poll it until it is granted (cleared) by hardware or optionally software could wait ~100 μs (guaranteed time for any associated DMA cycles to complete). In addition, software should also ensure that the receive packet buffer does not contain any residual packets of the same flow. See Section 4.6.7.1 for the required software flow. Only then the software can unlock the user buffers and re-use the context for a new FC exchange.

7.13.3.3.11 Invalidating FC Write Context

During nominal activity, hardware invalidates autonomously the FC contexts. The initiator indicates a completion of a granted portion of an FC write by sending a data frame with active sequence initiative flag. After receiving this type of frame, hardware invalidates the matched FC context. The header of this frame is posted to the legacy Rx queues with appropriate status indication. Hardware identifies this frame by the following criteria:

- The frame is identified as FCoE frame by its Ethernet type
- R_CTL -> Information (least significant four bits) equals 0x1 (solicited data)
- R_CTL -> Routing (most significant four bits) equals 0x0 (device data)
- F_CTL -> Sequence initiative equals 1b indicating transfer initiative to the target
- F_CTL -> End sequence” equals 1b indicating last frame in a sequence

Context that is invalidated autonomously by hardware is indicated by setting the *FCSTAT* field in the receive descriptor to 10b. When software gets this indication, it can unlock the user buffers instantly and re-use the context for a new FC exchange. If the FC write is not complete, software can re-use the same context for the completion of the exchange. It can also define a new user buffer list and indicate it to hardware by programming the DMA context. It then can enable the filter context by setting the *Re-Validation* bit the *WE* bit and the *FCoESEL* field in the FCFLTRW register.

Software can also invalidate a context in case of a time out event or other reasons. Software invalidation flow is described in Section 7.13.3.3.10.

7.13.3.3.12 OX_ID and RX_ID Pool Management

As previously indicated, hardware enables Large FC receive offload for up to 512 concurrent outstanding read or write requests. In some cases more than 512 concurrent outstanding requests are generated by the FCoE stack. Therefore, software would need to manage two separate queues for the requests: one queue for those FC requested supported by the large FC receive offload and another one for those requests that do not gain the large FC receive offload. Software should claim an entry in the context table, and its associated OX_ID or RX_ID for the duration of the read or write requests, respectively. Once a request completes and its context is invalidated, software can re-use its context entry for a new request.

Table 7-97 defines an example for an OX_ID list that can be used for new FC read requests managed by software at initialization time and during run time. Similarly, this table could be helpful for write requests and their RX_ID or shared pool for both read and write requests.

Table 7-97 Software OX_ID Table

Init State of the OX_ID Table	Run-Time Events	Updated State of the OX_ID Table	Run-Time Events	Updated State of the OX_ID Table	Run-Time events	Updated State of the OX_ID Table
0	Software is using 50 OX_ID values supported by large FC receive.	50	The following FC read requests are completed (and released by software) in the following order: 44, 21, 9, 0.	50	Software is using additional 50 OX_ID values supported by large FC receive. The following FC read requests are completed (and released by software) in the following order: 75, 10, 38. Ordering between software requests and releases does not matter in this example.	100
1		51		51		101
. . .		. . .		. . .		. . .
. . .		. . .		. . .		510
. . .		. . .		. . .		511
. . .		510		510		44
. . .		511		511		21
. . .				44		9
. . .				21		0
. . .				9		75
510				0		10
511						38

SW Note: Software is aware of which read requests can be offloaded by the large FC receive and use OX_IDs in the hardware range (0 to 511) only for those ones.

7.13.3.3.13 Packets and Headers Indication in the Legacy Receive Queue

The following packets or packets' headers are posted to the legacy receive queues:

- All FCoE frames that are not offloaded by the DDP logic
- Any packet with exception errors as described in Section 7.13.3.3.14
- Headers of packets posted to the user buffers by the DDP logic that contain meaningful data (as detailed in Section 7.13.3.3.2 and Section 7.13.3.3.3)

There are a few new fields in the receive descriptor dedicated to FCoE described in Section 7.1.6.2:

- Packet Type — FCoE packets are identified by their Ethernet type that is programmed in the ETQF registers.
- FCoE_PARAM — Reflects the value of the PARAM field in the DDP context.
- FCSTAT — FCoE DDP context indication.
- FCERR — FCoE Error indication. DDP offload is provided only when no errors are found.
- FCEOFs and FCEOFe — Status indication on the EOF and SOF flags in the Rx packet.

7.13.3.3.14 Exception Handling

Table 7-98 lists the exception errors related to FC receive functionality. Packets with any of the following exception errors are posted to the legacy receive queues with no DDP unless specified differently. In these cases, the exception error is indicated in the *Extended Error* field in the receive descriptor. The exceptions are listed in priority order in the table with highest priority first. Other then the EOF exception, any high priority exception hides all other ones with a lower priority.

Table 7-98 Exception Error Table

Event Description	Actions and Indications

Unsupported FCoE version (Rx Version > FCRXCTRL.FCOEVER)	The packet is identified as an FCoE packet type. DDP context parameters are left intact. Speculative CRC check is done. The packet is posted to legacy Rx queue regardless of CRC correctness (independent of FCRXCTRL.SavBad setting). If the packet matches the FCoE redirection table, the packet is posted to Rx queue index defined by the FCRETA[0].
	RDESC.STATUS.FCSTAT = 00b.
	RDESC.ERRORS.FCERR = 100b.

Incorrect FC CRC (see note 1).	Increment bad FC CRC count. FC context parameters are left intact. The packet can be posted to the legacy receive queues only if the FCRXCTRL.SavBad is set. If the packet matches the FCoE redirection table, the packet is posted to Rx queue index defined by the FCRETA[0].
	RDESC.STATUS.FCSTAT = 00b.
	RDESC.ERRORS.FCERR = 001b.

Rx packet with ESP option header.	If it matches the DDP context then auto invalidate the filter context while keeping the parameters intact. Note that this exception is not expected since software should not enable a context to an exchange that uses ESP encapsulation.
	RDESC.STATUS.FCSTAT = 00b / 01b / 10b.
	RDESC.ERRORS.FCERR = 000b.

Received EOFa or EOFni or any unrecognized EOF or SOF flags.	If it matches the DDP context then auto invalidate the filter context while keeping the parameters intact.
	RDESC.STATUS.FCSTAT = 00b / 01b / 10b.
	RDESC.ERRORS.FCERR = 010b (even if no DDP match).
	RDESC.ERRORS.FCEOFe = 1b.
	RDESC.STATUS.FCEOFs = 1b.

Received non-zero abort sequence condition in FC read exchange.	If it matches the DDP context then auto invalidate the filter context while keeping the parameters intact.
	RDESC.STATUS.FCSTAT = 00b / 01b / 10b.
	RDESC.ERRORS.FCERR = 010b (even if no DDP match).

Out of order reception of packet that matches a DDP context (see note 2).	Auto invalidate the filter context while keeping the parameters intact.
	RDESC.STATUS.FCSTAT = 01b.
	RDESC.ERRORS.FCERR = 100b.

Received unexpected EOF / SOF:
1) New sequence ID and SOF is not SOFi.
2) Last packet in a sequence and EOF is not EOFt.
	No DDP while filter context is updated (if matched and other parameters are in order).
	RDESC.STATUS.FCSTAT = 00b. / 01b.
	RDESC.ERRORS.FCERR = 000b.
	RDESC.ERRORS.FCEOFe = 1b.
	RDESC.STATUS.FCEOFs = 0b.

Table 7-98 Exception Error Table (Continued)

Event Description	Actions and Indications

The DMA unit gets FCoE packets while it missed the packet that was marked as first by the filter unit (see note 3).	Filter context parameters are updated while DMA context parameters are left intact.
	RDESC.STATUS.FCSTAT = 01b / 10b / 11b.
	RDESC.ERRORS.FCERR = 011b or 101b.

Last user buffer is exhausted (not enough space for the FC payload).	The filter context is updated while DMA context is auto invalidated.
	RDESC.STATUS.FCSTAT = 01b / 10b / 11b.
	RDESC.ERRORS.FCERR = 101b.

Legacy receive queue is not enabled or no legacy receive descriptor.	The entire packet is dropped. Auto invalidates the DMA context while the filter context remains active and continues to be updated regularly. Once legacy descriptors become valid again, packets are posted to the legacy queues with the following indication.
	RDESC.STATUS.FCSTAT = 01b / 10b / 11b.
	RDESC.ERRORS.FCERR = 101b.

Packet missed (lost) by the Rx packet buffer. Normally a case when flow control is not enabled or flow control does not work properly.	The entire packet is dropped (by the Rx packet buffer). Auto invalidate the DMA context while the filter context remains active and continues to be updated regularly. Once the Rx packet buffer gets free, further Rx packets are posted to the legacy queues with the following indication.
	RDESC.STATUS.FCSTAT = 00b / 01b / 10b / 11b.
	RDESC.ERRORS.FCERR = 110b.
	The software might ignore this error when FCSTAT equals 00b.

Note (1): Out of order might be one of the following cases. SEQ_CNT does not meet expected value. The *PARAM* field in the Rx packet does not match the DDP context. SEQ_ID keeps the same value as the previous packet in a new sequence identified by the presence of SOFi code in the *SOF* field.

Note (2): Lost sync between Filter and DMA contexts could be a result of context invalidation by software together with misbehaved target that sends packet with no host request.

7.13.3.3.15 FC Exchange Completion Interrupt

One of the performance indicators of an initiator is measured by the number of I/O operations per second it can generate. The number of FC exchanges per second is affected mainly by the CPU overhead associated with the FC exchange processing and software latencies. The number of concurrent outstanding FC exchanges supported by large FC receive is limited by hardware resources. Reducing the latency associated with processing completions increases the number of FC exchanges per second that the system supports.

The 82599 enables LLI for FCP_RSP frames or last FC data frame in a sequence with active *Sequence Initiative* flag. Any such frames can generate an LLI interrupt if the *FCOELLI* bit in the FCRXCTRL register is set.

Similarly, reducing the latency associated with processing FC write exchange can increase responder performance. During an FC write exchange, the originator handles the initiative to the responder after it sends all the data that the responder is ready to receive. Therefore, the 82599 enables LLI after receiving the last packet in a sequence with the *Sequence Initiative* bit set in the F_CTL field. The LLI is enabled by the same *FCOELLI* bit in the FCRXCTRL register previously indicated.

7.14 Reliability

7.14.1 Memory Integrity Protection

All the 82599 internal memories are protected against soft errors. Most of them are covered by ECC that correct single error per memory line and detect double errors per memory line. Few of the smaller memories are covered by parity protection that detects a single error per memory line.

Single errors in memories with ECC protection are named also as correctable errors. Such errors are silently corrected. Two errors in memories with ECC protection or single error in memories with parity protection are also named as un-correctable errors. Un-correctable errors are considered as fatal errors. If an un-correctable error is detected in Tx packet data, the packet is transmitted with a CRC error. If un-correctable error is detected in Rx packet data, the packet is reported to the host (or manageability) with a CRC error. If an un-correctable error is detected anywhere else, the 82599 halts the traffic and sets the ECC error interrupt. Software is then required to initiate a complete initialization cycle to resume nominal operation.

7.14.2 PCIe Error Handling

For PCIe error events and error reporting see Section 3.1.7.

8.0 Programming Interface

8.1 Address Regions

The 82599's address space is mapped into four regions with the PCI-Based Address Registers (BARs) listed in Table 8-1 and explained more in Section 9.3.6.1 and Section 9.3.6.2.

Table 8-1 the 82599 Address Regions

Addressable Content	Mapping Style	Region Size
Internal registers memories and Flash (memory BAR)	Direct memory mapped	128 KB + Flash_Size
Flash (optional)[1]	Direct memory-mapped	64 KB to 8 MB
Expansion ROM (optional)[2]	Direct memory-mapped	64 KB to 8 MB
Internal registers and memories (optional)[2]	I/O window mapped	32 bytes
MSI-X (optional)	Direct memory mapped	16 KB

1. The Flash space in the memory CSR and expansion ROM base address map is the same Flash memory. Accessing the memory BAR at offset 128 KB and expansion ROM at offset 0x0 are mapped to the Flash device at offset 0x0.
2. The internal registers and memories can be accessed though I/O space as explained in the sections that follow.

8.1.1 Memory-Mapped Access

8.1.1.1 Memory-Mapped Access to Internal Registers and Memories

The internal registers and memories can be accessed as direct memory-mapped offsets from the memory CSR BAR. See the following sections for detailed descriptions of the Device registers.

In IOV mode, this area is partially duplicated per Virtual Function (VF). All replications contain only the subset of the register set that is available for VF programming.

8.1.1.2 Memory-Mapped Accesses to Flash

The external Flash can be accessed using direct memory-mapped offsets from the CSR BAR (BAR0 in 32-bit addressing or BAR0/BAR1 in 64-bit addressing). The Flash is only accessible if enabled through the EEPROM Initialization Control word. For accesses, the offset from the CSR BAR minus 128 KB corresponds to the physical address within the external Flash device.

8.1.1.3 Memory-Mapped Access to MSI-X Tables

The MSI-X tables can be accessed as direct memory-mapped offsets from BAR3. The MSIX registers are described in Section 8.2.3.6.

In IOV mode, this area is duplicated per VF.

8.1.1.4 Memory-Mapped Access to Expansion ROM

The external Flash can also be accessed as a memory-mapped expansion ROM. Accesses to offsets starting from the expansion ROM base address reference the Flash, provided that access is enabled through the EEPROM Initialization Control word, and if the expansion ROM base address register contains a valid (non-zero) base memory address.

8.1.2 I/O-Mapped Access

All internal registers and memories can be accessed using I/O operations. I/O accesses are supported only if an I/O base address is allocated and mapped (BAR2), the BAR contains a valid (non-zero value), and I/O address decoding is enabled in the PCIe configuration.

When an I/O BAR is mapped, the I/O address range allocated opens a 32-byte window in the system I/O address map. Within this window, two I/O addressable registers are implemented: IOADDR and IODATA. The IOADDR register is used to specify a reference to an internal register or memory, and then the IODATA register is used to access it at the address specified by IOADDR:

Offset	Abbreviation	Name	RW	Size
0x0	IOADDR	Internal Register, Internal Memory, or Flash Location Address. 0x00000-0x1FFFF – Internal registers/memories. 0x20000-0x7FFFF – Undefined.	RW	4 bytes
0x04	IODATA	Data field for reads or writes to the internal register, internal memory, or Flash Location as identified by the current value in IOADDR. All 32 bits of this register are read/write capable.	RW	4 bytes
0x08-0x1F	Reserved	Reserved.	O	4 bytes