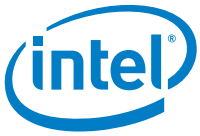


RXE

The RXE error bit is an indication for any MAC error. It is a logic or function of the following errors:

- CRC or symbol error might be a result of receiving a /V/ symbol on the TBI interface, /FE/ symbol on the GMII/XGMII interface, RX_ER assertion on GMII interface, bad EOP or loss of sync during packet reception.
- Undersize frames shorter than 64 bytes.
- Oversize frames larger than the MFS definition in the MAXFRS register.
- Length error in 802.3 packet format. Packets with an RXE error are posted to host memory only when store bad packet bit (FCTRL.SBP) is set.

VLAN Tag Field (16-bit offset 48, 2nd line)

If the RXDCTL.VME is set and the received packet type is 802.1q (as defined by VLNCTRL.VET) then the VLAN header is stripped from the packet data storage. In this case the 16 bits of the VLAN tag, priority tag and CFI from the received packet are posted to the *VLAN Tag* field in the receive descriptor. Otherwise, the *VLAN Tag* field contains 0x0000.

Table 7-14 VLAN Tag Field Layout (for 802.1q Packet)

15 13	12	11 0
PRI	CFI	VLAN

Priority and CFI are part of 803.1Q specifications. The VLAN field is provided in network order.

7.1.6 Advanced Receive Descriptors

7.1.6.1 Advanced Receive Descriptors — Read Format

Table 7-15 lists the advanced receive descriptor programming by the software. The SRRCTL[n]. DESCTYPE should be set to a value other than 000 when using the advanced descriptor format.

Table 7-15 Descriptor Read Format

	63 1	0
0	Packet Buffer Address [63:1]	A0
8	Header Buffer Address [63:1]	DD

Packet Buffer Address (64)

This is the physical address of the packet buffer. The lowest bit is A0 (LSB of the address).Header Buffer Address (64)

The physical address of the header buffer with the lowest bit being Descriptor Done (DD). When a packet spans in multiple descriptors, the header buffer address is used only on the first descriptor. During the programming phase, software must set the *DD* bit to zero (see the description of the DD bit in this section). This means that header buffer addresses are always word aligned.

When a packet spans in more than one descriptor, the header buffer address is not used for the second, third, etc. descriptors; only the packet buffer address is used in this case.

Note: The 82599 does not support null descriptors meaning packet or header addresses are zero.

7.1.6.2 Advanced Receive Descriptors — Write-Back Format

When the 82599 writes back the descriptors, it uses the format listed in Table 7-16. The SRRCTL[n]. DESCTYPE should be set to a value other than 000 when using the advanced descriptor format.

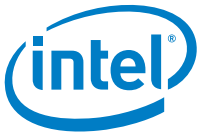
Table 7-16 Descriptor Write-Back Format

	63 48	47 32	31	30 21	20 17	16 4	3 0
0	RSS Hash / Fragment Checksum / RTT / FCoE_PARAM / Flow Director Filters ID		SPH	HDR_LEN	RSCCNT	Packet Type	RSS Type
8	VLAN Tag	PKT_LEN	Extended Error		Extended Status / NEXTP		
	63 48	47 32	31 20		19 0		

RSS Type (4-bit offset 0, 1st line)

The 82599 must identify the packet type and then choose the appropriate RSS hash function to be used on the packet. The RSS type reports the packet type that was used for the RSS hash function.

RSS Type	Description
0x0	No hash computation done for this packet
0x1	HASH_TCP_IPv4
0x2	HASH_IPv4
0x3	HASH_TCP_IPv6
0x4	Reserved
0x5	HASH_IPv6
0x6	Reserved
0x7	HASH_UDP_IPv4
0x8	HASH_UDP_IPv6



RSS Type	Description
0x9 – 0xE	Reserved
0xF	Packet reports flow director filters status

Packet Type (13-bit at offset 4, 1st line)

The Packet Type field reports the packet type identified by the hardware as follows. Note that some of the fields in the receive descriptor are valid for specific packet types. For example, the *FCOE_PARAM* field (multiplexed with the RSS) is valid only for FCoE packets.

Bit Index	Bit 11 = 0	Bit 11 = 1 (L2 packet)
0	IPV4 — IPv4 header present	Ethertype — ETQF register index that matches the packet. Special types are defined for 802.1x, 1588, and FCoE.
1	IPV4E — IPv4 with extensions	
2	IPV6 — IPv6 header present	
3	IPV6E — IPv6 with extensions	Reserved for extension of the *EtherType* field.
4	TCP — TCP header present	Reserved for extension of the *EtherType* field.
5	UDP — UDP header present	Reserved
6	SCTP — SCTP header	Reserved
7	NFS — NFS header	Reserved
8	IPSec ESP — IPSec encapsulation	Reserved
9	IPSec AH — IPSec encapsulation	Reserved
10	LinkSec — LinkSec encapsulation	LinkSec — LinkSec encapsulation
11	0b = non L2 packet	1b = L2 packet
12	Reserved	Reserved

Note: UDP, TCP and IPv6 indications are not set in an IPv4 fragmented packet.

In IOV mode, packets might be received from other local VMs. the 82599 does not check the L5 header for these packets and does not report NFS header. If such packets carry IP tunneling (IPv4 — IPv6), they are reported as IPV4E. The packets received from local VM are indicated by the *LB* bit in the status field.

RSC Packet Count- RSCCNT (4-bit offset 17, 1st line)

The *RSCCNT* field is valid only for RSC descriptors while in non-RSC it equals zero. RSCCNT minus one indicates the number of coalesced packets that start in this descriptor. RSCCNT might count up to 14 packets. Once 14 packets are coalesced in a single buffer, RSC is closed enabling accurate coalesced packet count. If the *RSCCNTBP* bit in RDRXCTL is set, coalescing might proceed beyond the 14 packets per buffer while RSCCNT stops incrementing beyond 0xF.

Note: Software can identify RSC descriptors by checking the *RSCCNT* field for non-zero value.

HDR_LEN (10-bit offset 21, 1st line)

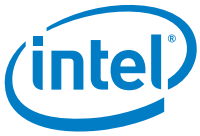
The HDR_LEN reflects the size of the packet header in byte units (if the header is decoded by the hardware). This field is meaningful only in the first descriptor of a packet and should be ignored in any subsequent descriptors. Header split is explained in Section 7.1.10 while the packet types for this functionality are enabled by the PSRTYPE[n] registers.

Split Header — SPH (1-bit offset 31, 1st line)

When set to 1b, indicates that the hardware has found the length of the header. If set to 0b, the header buffer may be used only in split always mode. The header buffer length as well as split header support is indicated in the following table. If the received header size is greater or equal to 1024 bytes, the *SPH* bit is not set and header split functionality is not supported. The *SPH* bit is meaningful only on the first descriptor of a packet. See additional details on SPH, PKT_LEN and HDR_LEN as a function of split modes in Table 7-20.

Packet Type	Header Length (includes all fields up to the field specified)	Header Split
Unrecognized Ethertype only with / without SNAP and with / without VLAN or packets that match the L2 filters (MTQF) other than FCoE with / without VLAN.	VLAN header(s) if present Else, *EtherType* field	No
FCoE packet without ESP option header	FC header including FC options	N/A (1)
FCoE packet with ESP option header	FC header excluding FC options	N/A (1)
IPv4 only or fragmented IPv4 with any payload including IPv4-IPv6 tunneling	IPv4 header	Enabled
Non-fragmented IPv4, TCP / UDP / SCTP	L4 header	Enabled
IPv4-IPv6, only or fragmented IPv4-IPv6 at IPv6 header with any payload	IPv6 header (up to the fragment extension header if exist)	Enabled
IPv4-IPv6,TCP / UDP / SCTP	L4 header	Enabled
IPv4 / IPv6 / IPv4-IPv6, TCP / UDP, NFS	L5 header	Enabled

Notes: (1) Header split is not permitted in queues that might receive FCoE packets.



RSS Hash or FCOE_PARAM or Flow Director Filters ID (32-bit offset 32, 1st line) / Fragment Checksum (16-bit offset 48, 1st line)

This field has multiplexed functionality according to the received packet type (reported on the *Packet Type* field in this descriptor) and device setting.

FCoE_PARAM

For FCoE packets that matches a valid DDP context, this field holds the *PARAM* field in the DDP context after processing the received packet. If the *Relative Offset Present* bit in the F_CTL was set in the data frames, the *PARAM* field indicates the size in bytes of the entire exchange inclusive the frame reported by this descriptor.

Fragment Checksum

For non-FCoE packets, this field might hold the UDP fragment checksum (described in Section 7.1.13) if both the RXCSUM.PCSD bit is cleared and RXCSUM. The *IPPCSE* bit is also set. This field is meaningful only for UDP packets when the *UDPV* bit in the Extended Status word is set.

RSS Hash / Flow Director Filters ID

For non-FCoE packets, this field might hold the RSS hash value or flow director filters ID if the RXCSUM.PCSD bit is set. Furthermore, if the FDIRCTRL.Report-Status bit is set, then the flow director filters ID is reported; otherwise, the RSS hash is reported.

RSS Hash

The RSS hash value is required for RSS functionality as described in Section 7.1.2.8.

Flow Director Filters ID

The flow director filters ID is reported only when the received packet matches a flow directory filter (see Section 7.1.2.7).

The flow director filter ID field has a different structure for signature-based filters and perfect match filters as follows:

Filter Type	31	30 29	28 16	15 13	12 0
Hash-based Flow Director Filter ID	Rsv	Bucket Hash		Signature	
Perfect Match Flow Director Filter ID	Rsv		Hash	Rsv	SW-Index

Bucket Hash

A hash value that identifies a flow director bucket. When the flow director table is smaller than 32 K filters the bucket hash is smaller than 15 bits. In this case the upper bit(s) are set to zero.

Signature

A hash value used to identify flow within a bucket.

SW-Index

The SW-Index that is taken from the filter context, programmed by software. It is meaningful only when the *FLM* bit in the Extended Status is set as well.

Rsv

Reserved.

Extended Status / NEXTP (20-bit offset 0, 2nd line)

Status information indicates whether the descriptor has been used and whether the referenced buffer is the last one for the packet. Table 7-17 lists the extended status word in the last descriptor of a packet (*EOP* bit is set). Table 7-18 lists the extended status word in any descriptor but the last one of a packet (*EOP* bit is cleared).

Table 7-17 Receive Status (RDESC.STATUS) Layout of *Last* Descriptor

19	18	17	16	15	14	13	12	11	10
Rsv	LB	SECP	TS	Rsv				LLINT	UDPV

VEXT	Rsv	PIF	IPCS	L4I	UDPCS	VP	FLM	EOP	DD
			FCEOFs	FCSTAT					
9	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Table 7-18 Receive Status (RDESC.STATUS) Layout of *Non-Last* Descriptor

19 4	3 : 2	1	0
Next Descriptor Pointer — NEXTP	Rsv	EOP = 0b	DD

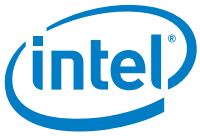
Rsv (8), Rsv (15:12), Rsv(19) — Reserved at zero.

FLM(2) — Flow director filter match indication is set for packets that match these filters.

VP(3), PIF (7) — These bits are described in the legacy descriptor format in Section 7.1.5.

EOP (1) and DD (0) — *End of Packet* and *Done* bits are listed in the following table:

DD	EOP	Description
0	0	Software setting of the descriptor when it hands it to hardware.
0	1	Reserved (invalid option)
1	0	A completion status indication for a non last descriptor of a packet (or multiple packets in the case of RSC) that spans across multiple descriptors. In a single packet case the *DD* bit indicates that the hardware is done with the descriptor and its buffers. In the case of RSC, the *DD* bit indicates that the hardware is done with the descriptor but might still use its buffers (for the coalesced header) until the last descriptor of the RSC completes. Only the *Length* fields are valid on this descriptor. In the RSC case, the next descriptor pointer is valid as well.
1	1	A completion status indication of the entire packet (or the multiple packets in the case of RSC) and software might take ownership of its descriptors. All fields in the descriptor are valid (reported by the hardware).



UDPCS(4), L4I (5) / FCSTAT (5:4) — This field has multiplexed functionality for FCoE and non-FCoE packets. Hardware identifies FCoE packets in the filter unit and indicates it in the *Packet Type* field in the Rx descriptor. For non-FCoE packets this field is UDPCS and L4I. The UDPCS (UDP checksum) is set when hardware provides UDP checksum offload. The L4I (L4 Integrity) is set when hardware provides any L4 offload as: UDP checksum, TCP checksum or SCTP CRC offload. For FCoE packets, this field represents the FCSTAT (FCoE Status) as follows:

FCSTAT	Meaning
00	No match to any active FC context
01	FCoE frame matches an active FC context with no DDP. The entire frame is posted to the receive buffer indicated by this descriptor.
10	FCP_RSP frame received that invalidates an FC read context or last data packet in a sequence with sequence initiative set that invalidates an FC write context.
11	FCoE frame matches an active FC context and found liable for DDP by the filter unit. The packet's data was posted directly to the user buffers if no errors were found by the DMA unit as reported in the *FCERR* field. If any error is found by the DMA unit the entire packet is posted to the legacy queues.

IPCS(6), FCEOFs (6) — This bit has multiplexed functionality for FCoE and non-FCoE packets. The hardware identifies FCoE packets in the filter unit and indicates it in the *Packet Type* field in the Rx descriptor. For non-FCoE packets it is IPCS as described in Legacy Rx descriptor (in Section 7.1.5). For FCoE packets, this bit and the *FCEOFe* bit in the *Extended Error* field indicates the received EOF code as follows:

FCEOFe	FCEOFs	Description and Digested meaning and Device Behavior
0	0	EOFn. Nominal operation, DDP is enabled.
0	1	EOFt. Nominal operation (end of sequence), DDP is enabled.
1	0	Unexpected EOFn-EOFt or SOFi-SOFn. No DDP while filter context is updated by the packet.
1	1	EOFa, EOFni or un-recognized EOF / SOF. No DDP while filter context is invalidated.

VEXT (9) — Outer-VLAN is found on a double VLAN packet. This bit is valid only when CTRL_EXT.EXTENDED_VLAN is set. See more details in Section 7.4.5.

UDPV (10) — The *UDP Checksum Valid* bit indicates that the incoming packet contains a valid (non-zero value) checksum field in an incoming fragmented (non-tunneled) UDP IPv4 packet. It means that the *Fragment Checksum* field in the receive descriptor contains the UDP checksum as described in Section 7.1.13. When this field is cleared in the first fragment that contains the UDP header, it means that the packet does not contain a valid UDP checksum and the checksum field in the Rx descriptor should be ignored. This field is always cleared in incoming fragments that do not contain the UDP header.

LLINT (11) — The *Low Latency Interrupt* bit indicates that the packet caused an immediate interrupt via the low latency interrupt mechanism.

TS (16) — The *Time Stamp* bit is set when the device recognized a time sync packet. In such a case the hardware captures its arrival time and stores it in the Time Stamp register. For more details see Section 7.9.

SECP (17) — Security processing bit indicates that the hardware identified the security encapsulation and processed it as configured.

LinkSec processing — This bit is set each time LinkSec processing is enabled regardless if a matched SA was found.

IPsec processing — This bit is set only if a matched SA was found. Note that hardware does not process packets with an IPv4 option or IPv6 extension header and the SECP bit is not set. This bit is not set for IPv4 packets shorter than 70 bytes, IPv6 ESP packets shorter than 90 bytes, or IPv6 AH packets shorter than 94 bytes (all excluding CRC). Note that these packet sizes are never expected and set the length error indication in the SECERR field.

LB (18) — This bit provides a loopback status indication which means that this packet is sent by a local VM (VM to VM switch indication).

NEXTP (19:4) — Large receive might be composed of multiple packets and packets might span in multiple buffers (descriptors). These buffers are not guaranteed to be consecutive while the *NEXTP* field is a pointer to the next descriptor that belongs to the same RSC. The *NEXTP* field is defined in descriptor unit (the same as the head and tail registers). The *NEXTP* field is valid for any descriptor of a large receive (the *EOP* bit is not set) except the last one. It is valid even in consecutive descriptors of the same packet. In the last descriptor (on which the *EOP* bit is set), *NEXTP* is not indicated but rather all other status fields previously described in this section.

Extended Error (12-bit offset 21, 2nd line)

Table 7-19 and the following text describe the possible errors reported by hardware.

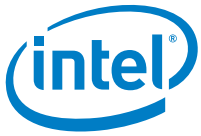
t

Table 7-19 Receive Errors (RDESC.ERRORS) Layout

11	10	9	8:7	6:4	3	2:0
IPE	L4E	RXE	Rsv	Rsv	HBO	Rsv
FCEOFe			SECERR			FCERR / FDIRERR

FCERR (2:0) — Defines error cases for FCoE packets. Note that hardware indicates FCoE packet recognition on the Packet Type field in the Rx descriptor. Packets with FCERR are posted to host memory regardless of the store bad packet setting in the Filter Control register.

FCERR Code	Meaning
000	No exception errors found
001	Bad FC CRC. Hardware does not check any other FC fields in the packet.
010	One of the following error indications found by the filter unit (hardware auto-invalidates a matched DDP filter context if exists): 1. Received non-zero abort sequence condition in the *F_CTL* field in FC read packet. 2. Received EOFa or EOFni or any un-recognized EOF or SOF flags.
011	The DMA unit gets FCoE packets that match a DDP context while it missed the packet that was marked as first by the filter unit. Filter context parameters might be updated while DMA context parameters are left intact (see error code 101b).



FCERR Code	Meaning
100	One of the following cases: 1. Unsupported FCoE version. FCSTAT equals to 00b. 2. Out of order reception (SEQ_CNT does not match expected value) of a packet that matches an active DDP context. The filter unit might set the FCSTAT to 01b, 10b or 11b.
101	No DMA resources due to one of the following cases listed while the hardware auto-invalidates the DDP DMA context. Software should invalidate the filter context before it can reuse it. (1) Last buffer exhausted (no space in the user buffers). (2) Legacy receive queue is not enabled or no legacy receive descriptor. (3) Some cases of a missed packet as described in FCERR code 011b. This code should be ignored when FCSTAT equals 00b (meaning no context match).
110	Filter context valid and DMA context invalid. Indicates that some packet(s) were lost by the DMA context due to lack of legacy receive descriptors or were missed by the Rx packet buffer. *Note:* The software might ignore this error when FCSTAT equals 00b.
111	Reserved

FDIRERR (2:0) — This field is relevant for non-FCoE packets when the flow director filters are enabled.

FDIRErr(0) - Length — If the flow director filter matches the *Length* bit, this indicates that the distance of the matched filter from the hash table exceeds the FDIRCTRL.Max-Length. If there is no matched filter, the *Length* bit is set if the flow director linked list of the matched hash value exceeds the FDIRCTRL.Max-Length.

FDIRErr(1) - Drop — The *Drop* bit indicates that a received packet matched a flow director filter with a drop action. In the case of perfect mode filtering, it is expected to find the drop indication only when the linked list in the flow director bucket exceeds the permitted Max-Length. In this case, the packet is not dropped. Instead, it is posted to the Rx queue (indicated in the filter context) for software handling of the Max-Length exception. In the case of hash mode filtering, it is expected that the drop queue is always a valid queue so all packets that match the drop filter are visible to software.

FDIRErr(2) - Coll — A matched flow director filter with a collision indication was found. The collision indicates that software attempted to step over this filter with a different action that was already programmed.

HBO (3) — The *Header Buffer Overflow* bit is set if the packet header (calculated by hardware) is bigger than the header buffer (defined by PSRCTL.BSIZEHEADER). *HBO* reporting might be used by software to allocate bigger buffers for the headers. It is meaningful only if the *SPH* bit in the receive descriptor is set as well. The HDR_LEN field is valid even when the *HBO* bit is set. Packets with HBO error are posted to host memory regardless of the store bad packet setting (FCTRL.SBP). Packet DMA to its buffers when the *HBO* bit is set, depends on the device settings as follows:

SRRCTL.DESCTYPE	DMA Functionality
Header Split (010b)	The header is posted with the rest of the packet data to the packet buffer.
Always Split Mode (101b)	The header buffer is used as part of the data buffers and contains the first PSRCTL.BSIZEHEADER bytes of the packet.

Rsv (5:4) — Reserved at zero.

SECERR (8:7) — Security error indication for LinkSec or IPsec. This field is meaningful only if the SECP bit in the extended status is set.

SECERR	LinkSec Error Reporting	IPsec Error Reporting
00	No errors found or no security processing	No errors found while an active SA found or no security processing.
01	No SA match	Invalid IPsec Protocol: IPsec protocol field (*ESP* or *AH*) in the received IP header does not match expected one stored in the SA table.
10	Replay error	Length error: ESP packet is not 4-bytes aligned or AH/ESP header is truncated (for example, a 28-byte IPv4 packet with IPv4 header + ESP header that contains only SPI and SN) or *AH Length* field in the AH header is different than 0x07 for IPv4 or 0x08 for IPv6 or the entire packet size excluding CRC is shorter than 70 bytes for IPv4 or 90 bytes for IPv6 ESP or 94 bytes for IPv6 AH.
11	Authentication failed: Bad signature	Authentication failed: Bad signature.

RXE (9) — RXE is described in the legacy descriptor format in Section 7.1.5.

L4E (10) — L4 integrity error is valid only when the *L4I* bit in the *Status* field is set. It is active if L4 processing fails (TCP checksum or UDP checksum or SCTP CRC). Packets with L4 integrity error are posted to host memory regardless of the store bad packet setting (FCTRL.SBP).

FCEOFe(11) / IPE(11) — This bit has multiplexed functionality. FCoE packets are indicated as such in the Packet Type field in the Rx descriptor.

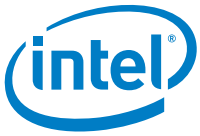
Non-FCoE Packet	FCoE Packet
IPE (IPv4 checksum error) is described in Section 7.1.5.	FC EOF Exception (**FCEOFe**). This bit indicates unexpected EOF or SOF flags. The specific error is defined by the FCEOF bit in the extended status previously described.

PKT_LEN (16-bit offset 32, 2nd line)

PKT_LEN holds the number of bytes posted to the packet buffer. The length covers the data written to a receive buffer including CRC bytes (if any). Software must read multiple descriptors to determine the complete length for packets that span multiple receive buffers. If SRRCTL.DESCTYPE = 2 (advanced descriptor header splitting) and the buffer is not split because the header is bigger than the allocated header buffer, this field reflects the size of the data written to the data buffer (header + data).

VLAN Tag (16-bit offset 48, 2nd line)

This field is described in the legacy descriptor format in Section 7.1.5.



7.1.7 Receive Descriptor Fetching

7.1.7.1 Fetch On Demand

The 82599 implements a fetch-by-demand mechanism for descriptor fetch. Descriptors are not fetched in advance, but rather fetched after a packet is received. Such a strategy eliminates the need to store descriptors on-die for each and every descriptor queue in anticipation for packet arrival.

7.1.8 Receive Descriptor Write-Back

The 82599 writes back the receive descriptor immediately following the packet write into system memory. It is therefore possible for a single descriptor to be written at a time into memory. However, if aggregation occurs during descriptor fetch (see Section 7.1.7), then the descriptors fetched in the aggregated operation are written back in a single write-back operation. In Receive Coalescing (RSC), all the descriptors except the last one are written back when they are completed. This does not have to be on packet boundaries but rather when the next descriptor of the same RSC is fetched. See Section 7.11.5.1 for more on RSC.

Note: Software can determine if a packet has been received by checking the receive descriptor *DD* bit in memory or by checking the value of the receive head pointer in the RDH/RDL registers. Checking through *DD* bits eliminates a potential race condition: all descriptor data is posted internally prior to incrementing the head register and a read of the head register could potentially pass the descriptor waiting inside the 82599.

7.1.9 Receive Descriptor Queue Structure

Figure 7-11 shows the structure of each of the receive descriptor rings. Note that each ring uses a contiguous memory space.

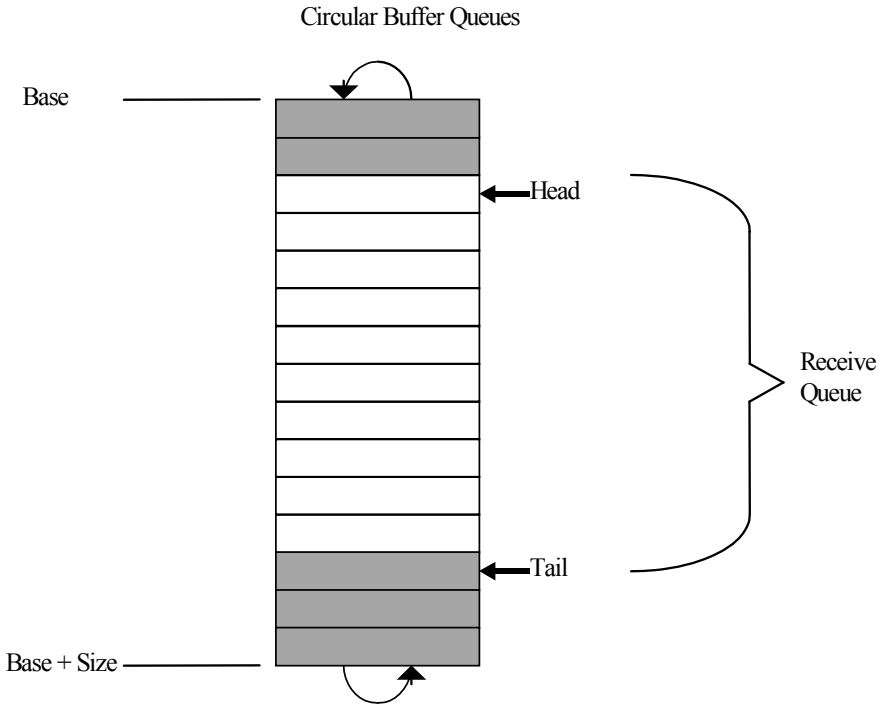


Figure 7-11 Receive Descriptor Ring Structure

Software inserts receive descriptors by advancing the tail pointer(s) to refer to the address of the entry just beyond the last valid descriptor. This is accomplished by writing the descriptor tail register(s) with the offset of the entry beyond the last valid descriptor. The 82599 adjusts its internal tail pointer(s) accordingly. As packets arrive, they are stored in memory and the internal head pointer(s) is incremented by the 82599.

When RSC is not enabled, the visible (external) head pointer(s) reflect the internal ones. On any receive queue that enables RSC, updating the external head pointer might be delayed until interrupt assertion. When the head pointer(s) is equal to the tail pointer(s), the queue(s) is empty. The 82599 stops storing packets in system memory until software advances the tail pointer(s), making more receive buffers available.

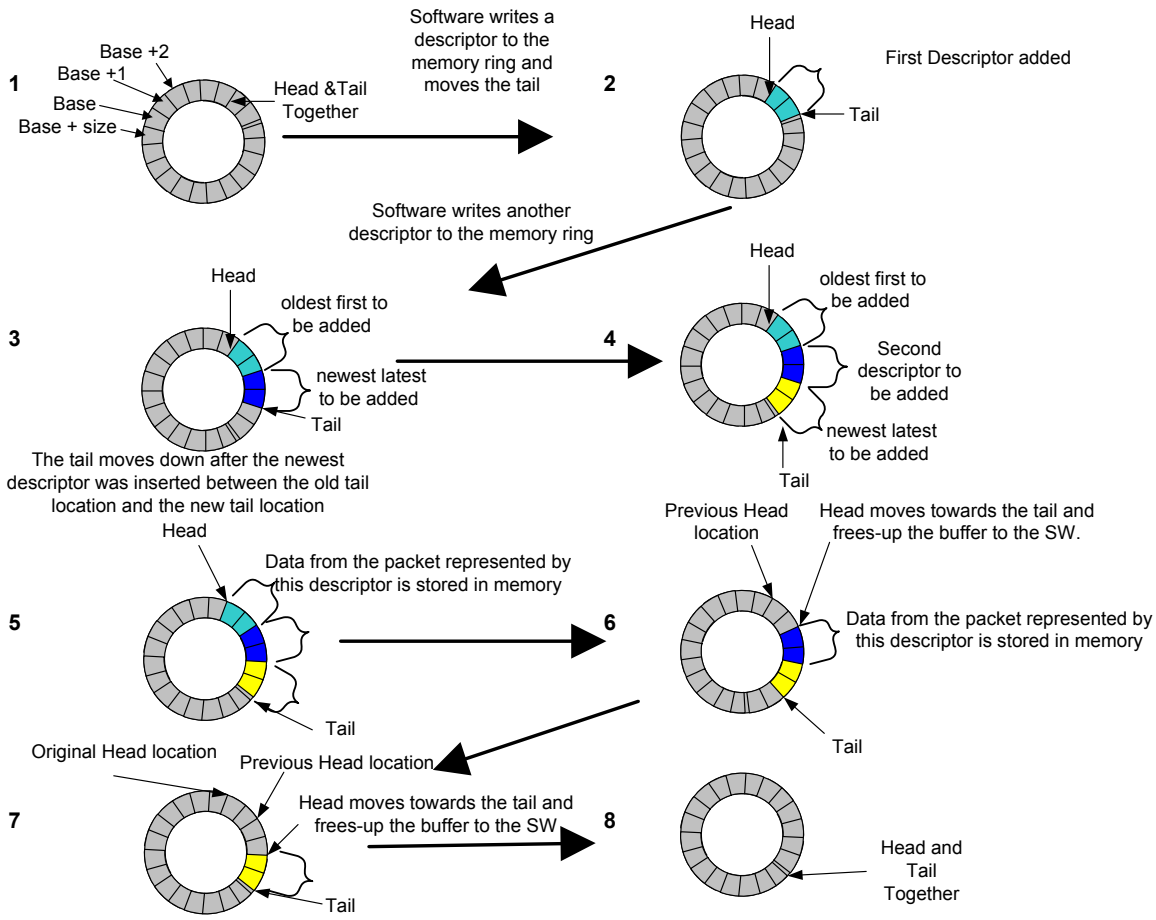


Figure 7-12 Descriptors and Memory Rings

The 82599 writes back used descriptors just prior to advancing the head pointer(s). Head and tail pointers wrap back to base when the number of descriptors corresponding to the size of the descriptor ring have been processed.

The receive descriptor head and tail pointers reference to 16-byte blocks of memory. Shaded boxes in Figure 7-12 represent descriptors that have stored incoming packets but have not yet been recognized by software. Software can determine if a receive buffer is valid by reading descriptors in memory rather than by I/O reads. Any descriptor with a *DD* bit set has been used by the hardware, and is ready to be processed by software.

Note: The head pointer points to the next descriptor that is to be written back. At the completion of the descriptor write-back operation, this pointer is incremented by the number of descriptors written back. Hardware owns all descriptors between [head... tail]. Any descriptor not in this range is owned by software.

The receive descriptor rings are described by the following registers:

- Receive Descriptor Base Address registers (RDBA) — This register indicates the start of the descriptor ring buffer; this 64-bit address is aligned on a 16-byte boundary and is stored in two consecutive 32-bit registers. Hardware ignores the lower 4 bits.
- Receive Descriptor Length registers (RDLEN) — This register determines the number of bytes allocated to the circular buffer. This value must be a multiple of 128 (the maximum cache line size). Since each descriptor is 16 bytes in length, the total number of receive descriptors is always a multiple of 8.
- Receive Descriptor Head registers (RDH) — This register holds a value that is an offset from the base, and indicates the in-progress descriptor. There can be up to 64K-8 descriptors in the circular buffer. Hardware maintains a shadow copy that includes those descriptors completed but not yet stored in memory.

  Software can determine if a packet has been received by either of two methods: reading the *DD* bit in the receive descriptor field or by performing a PIO read of the Receive Descriptor Head register. Checking the descriptor *DD* bit in memory eliminates a potential race condition. All descriptor data is written to the I/O bus prior to incrementing the head register, but a read of the head register could pass the data write in systems performing I/O write buffering. Updates to receive descriptors use the same I/O write path and follow all data writes. Consequently, they are not subject to the race.
- Receive Descriptor Tail registers (RDT) — This register holds a value that is an offset from the base, and identifies the location beyond the last descriptor hardware can process. This is the location where software writes the first new descriptor.

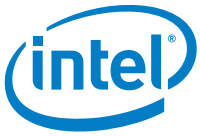
If software statically allocates buffers, and uses a memory read to check for completed descriptors, it simply has to zero the status byte in the descriptor to make it ready for re-use by hardware. This is not a hardware requirement, but is necessary for performing an in-memory scan. This is relevant only to legacy descriptors.

All the registers controlling the descriptor rings behavior should be set before receive is enabled, apart from the tail registers which are used during the regular flow of data.

7.1.9.1 Low Receive Descriptors Threshold

As previously described, the size of the receive queues is measured by the number of receive descriptors. During run time, software processes descriptors and upon completion of descriptors, increments the Receive Descriptor Tail registers. At the same time, the hardware may post new received packets incrementing the Receive Descriptor Head registers for each used descriptor.

The number of usable (free) descriptors for the hardware is the distance between the Tail and Head registers. When the tail reaches the head, there are no free descriptors and further packets might be either dropped or block the receive FIFO. In order to avoid this situation, the 82599 might generate a low latency interrupt (associated to the relevant Rx queue) once there are less equal free descriptors than specified by a low level threshold. The threshold is defined in 64 descriptors granularity per queue in the SRRCTL[n].RDMTS field.



7.1.10 Header Splitting

Note: Header Splitting mode might cause unpredictable behavior and should not be used with the 82599. For more information, see the product specification update errata on this subject.

7.1.10.1 Purpose

This feature consists of splitting a packet header to a different memory space. This helps the host to fetch headers only for processing: headers are posted through a regular snoop transaction in order to be processed by the host CPU. It is recommended to perform this transaction with DCA enabled (see Section 7.5).

The packet (header + payload) is stored in memory. Later, an IOAT transaction moves the payload from the driver space to the application memory.

The 82599's support for header split is controlled by the DESCTYPE field of the Split Receive Control registers (SRRCTL). The following modes exist in both split and non-split modes:

- 000b: Legacy mode - Legacy descriptors are used, headers and payloads are not split.
- 001b: Advanced mode, no split - Advanced descriptors are in use, header and payload are not split.
- 010b: Advanced mode, header split - Advanced descriptors are in use, header and payload are split to different buffers.
- 101b: Advanced mode, split always - Advanced descriptors are in use, header and payload are split to different buffers. If no split is done, the first part of the packet is stored in the header buffer. When using a split always descriptor type, the header buffer size (BSIZEHEADER) should be set to four which equals to 256 bytes.

The 82599 uses packet splitting when the SRRCTL[n].DESCTYPE is greater than one.

7.1.10.2 Description

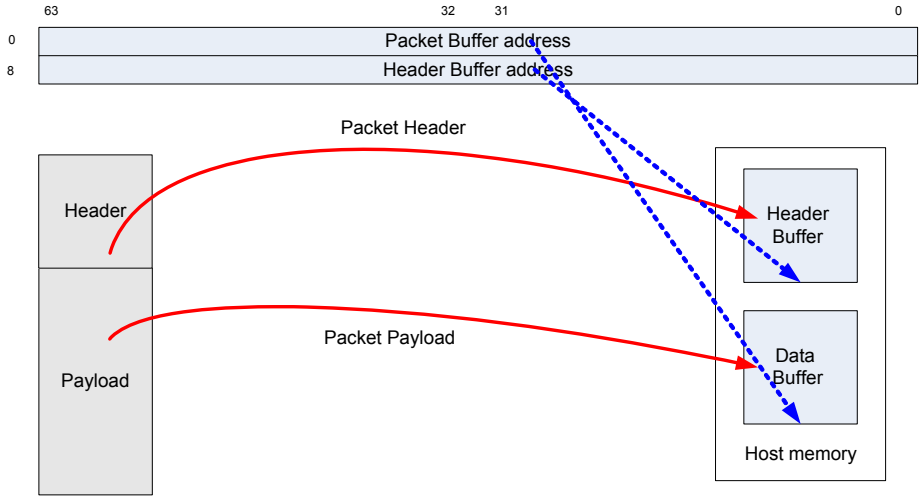


Figure 7-13 Header Splitting Diagram

The physical address of each buffer is written in the *Buffer Addresses* fields:

- The packet buffer address includes the address of the buffer assigned to the packet data.
- The header buffer address includes the address of the buffer that contains the header information. The receive DMA module stores the header portion of the received packets into this buffer.

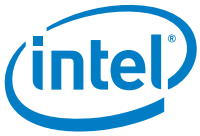
The sizes of these buffers are statically defined in the SRRCTL[n] registers:

- The *BSIZEPACKET* field defines the size of the buffer for the received packet.
- The *BSIZEHEADER* field defines the size of the buffer for the received header. If header split is enabled, this field must be configured to a non-zero value. The 82599 only writes the header portion into the header buffer. The header size is determined by the options enabled in the PSRTYPE registers.

When header split is selected, the packet is split only on selected types of packets. A bit exists for each option in PSRTYPE[n] registers, so several options can be used in conjunction. If one or more bits are set, the splitting is performed for the corresponding packet type. See Section 8.2.3.7.4 for details on the possible header types supported. In virtualization mode, a separate PSRTYPE register is provided per pool up to the number of pools enabled. In non-virtualization mode, only PSRTYPE[0] is used.

Rules regarding header split:

- Packets that have headers bigger than 1023 bytes are not split.
- The header of a fragmented IPv6 packet is defined until the fragmented extension header.
- Header split must not be used in a queue used for a FCoE large receive.
- An IP packet with more than a single IP header (such as any combination of IPv4 and IPv6 tunneling) is not split.



- Packet header cannot span across buffers, therefore, the size of the header buffer must be larger than any expected header size. In case of header split mode (SRRCTL.DESCTYPE = 010b), a packet with a header larger than the header buffer is not split.
- If an IPsec header is present in the receive packet, the following rules apply:
  - IPsec packets handled in the 82599 always include IPsec header in a split done at IP boundary.
  - IPsec packets handled in software must never do header split.

Table 7-20 lists the behavior of the 82599 in the different modes.

Table 7-20 Behavior in Header Split Modes

DESCTYPE	Condition	SPH	HBO	PKT_LEN	HDR_LEN	Header and Payload DMA
Split	1. Header can't be decoded	0	0	Min(packet length, buffer size)	0x0	Header + Payload --> Packet Buffer
	2. Header <= BSIZEHEADER	1	0	Min (payload length, buffer size)[3]	Header size	Header --> Header Buffer Payload --> Packet Buffer
	3. Header > BSIZEHEADER	1	1	Min (packet length, buffer size)	Header size[5]	Header + Payload --> Packet Buffer
Split – always use header buffer	1. Header can't be decoded and packet length <= BSIZEHEADER	0	0	0x0	Packet length	Header + Payload --> Header Buffer
	2. Header can’t be decoded and packet length > BSIZEHEADER	0	0	Min (packet length – BSIZEHEADER, data buffer size)	Undefined	Header + Payload --> Header + Packet Buffers[4]
	3. Header <= BSIZEHEADER	1	0	Min (payload length, data buffer size)	Header Size	Header --> Header Buffer Payload --> Packet Buffer
	4. Header > BSIZEHEADER	1	1	Min (packet length – BSIZEHEADER, data buffer size)	Header Size[5]	Header + Payload --> Header + Packet Buffer

Notes:

1. Partial means up to BSIZEHEADER.
2. HBO is set to 1b if the header size is bigger than BSIZEHEADER and zero otherwise.
3. In a header only packet (such as TCP ACK packet), the PKT_LEN is zero.
4. If the packet spans more than one descriptor, only the header buffer of the first descriptor is used.
5. HDR_LEN doesn't reflect the actual data size stored in the header buffer. It reflects the header size determined by the parser.

7.1.11 Receive Checksum Offloading

The 82599 supports the offloading of three receive checksum calculations: the fragment checksum, the IPv4 header checksum, and the TCP/UDP checksum.

For supported packet/frame types, the entire checksum calculation can be offloaded to the 82599. The 82599 calculates the IPv4 checksum and indicates a pass/fail indication to software via the *IPv4 Checksum Error* bit (RDESC.IPE) in the *ERROR* field of the receive descriptor. Similarly, the 82599 calculates the TCP or UDP checksum and indicates a pass/fail condition to software via the *TCP/UDP Checksum Error* bit (RDESC.TCPE). These error bits are valid when the respective status bits indicate the checksum was calculated for the packet (RDESC.IPCS and RDESC.L4CS, respectively).

Similarly, if RFCTL.Ipv6_DIS and RFCTL.IP6Xsum_DIS are cleared to zero, the 82599 calculates the TCP or UDP checksum for IPv6 packets. It then indicates a pass/fail condition in the *TCP/UDP Checksum Error* bit (RDESC.TCPE).

Supported frame types:

- Ethernet II
- Ethernet SNAP

Table 7-21 Supported Receive Checksum Capabilities

Packet Type	Hardware IP Checksum Calculation	Hardware TCP/UDP Checksum Calculation
IP header's protocol field contains a protocol # other than TCP or UDP.	Yes	No
IPv4 + TCP/UDP packets.	Yes	Yes
IPv6 + TCP/UDP packets.	No (N/A)	Yes
IPv4 packet has IP options (IP header is longer than 20 bytes).	Yes	Yes
IPv6 packet with next header options: • Hop-by-hop options. • Destinations options (without home address). • Destinations options (with home address). • Routing (with segment left 0). • Routing (with segment left > 0). • Fragment.	 No (N/A) No (N/A) No (N/A) No (N/A) No (N/A) No (N/A)	 Yes Yes No Yes No No
Packet has TCP or UDP options.	Yes	Yes
Ipv4 tunnels: • IPv4 packet in an IPv4 tunnel. • IPv6 packet in an IPv4 tunnel.	 No Yes (IPv4)	 No No
IPv6 tunnels: • IPv4 packet in an IPv6 tunnel. • IPv6 packet in an IPv6 tunnel.	 No No	 No No
Packet is an IPv4 fragment.	Yes	UDP checksum assist

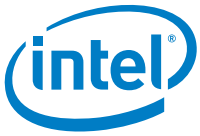


Table 7-21 Supported Receive Checksum Capabilities (Continued)

Packet Type	Hardware IP Checksum Calculation	Hardware TCP/UDP Checksum Calculation
Packet is greater than 1522 bytes.	Yes	Yes
Packet has 802.3ac tag.	Yes	Yes

The previous table lists general details about what packets are processed. In more detail, the packets are passed through a series of filters to determine if a receive checksum is calculated.

Ethernet MAC Address Filter

This filter checks the MAC destination address to be sure it is valid (that is IA match, broadcast, multicast, etc.). The receive configuration settings determine which Ethernet MAC addresses are accepted. See the various receive control configuration registers such as FCTRL, MCSTCTRL (RTCL.UPE, MCSTCTRL.MPE, FCTRL.BAM), MTA, RAL, and RAH for details.

SNAP/VLAN Filter

This filter checks the next headers looking for an IP header. It is capable of decoding Ethernet II, Ethernet SNAP, and IEEE 802.3ac headers. It skips past any of these intermediate headers and looks for the IP header. The receive configuration settings determine which next headers are accepted. See the various receive control configuration registers such as VLNCTRL.VFE, VLNCTRL.VET, and VFTA for more details.

IPv4 Filter

This filter checks for valid IPv4 headers. The version field is checked for a correct value (4).

IPv4 headers are accepted if they are any size greater than or equal to five (Dwords). If the IPv4 header is properly decoded, the IP checksum is checked for validity.

IPv6 Filter

This filter checks for valid IPv6 headers, which are a fixed size and have no checksum. The IPv6 extension headers accepted are: Hop-by-hop, destination options, and routing. The maximum extension header size supported is 256 bytes. The maximum total header size supported is 1 KB.

IPv6 Extension Headers

IPv4 and TCP provide header lengths, which enable hardware to easily navigate through these headers on packet reception for calculating checksum and CRC, etc. For receiving IPv6 packets, however, there is no IP header length to help hardware find the packet's ULP (such as TCP or UDP) header. One or more IPv6 extension headers might exist in a packet between the basic IPv6 header and the ULP header. Hardware must skip over these extension headers to calculate the TCP or UDP checksum for received packets.

The IPv6 header length without extensions is 40 bytes. The IPv6 field *Next Header Type* indicates what type of header follows the IPv6 header at offset 40. It might be an upper layer protocol header such as TCP or UDP (next header type of 6 or 17, respectively), or it might indicate that an extension header follows. The final extension header indicates with it's *Next Header Type* field the type of ULP header for the packet.

IPv6 extension headers have a specified order. However, destinations must be able to process these headers in any order. Also, IPv6 (or IPv4) might be tunneled using IPv6, and thus another IPv6 (or IPv4) header and potentially its extension headers might be found after the extension headers.

The IPv4 next header type is at byte offset 9. In IPv6, the first next header type is at byte offset 6.

All IPv6 extension headers have the next header type in their first eight bits. Most have the length in the second eight bits (Offset Byte[1]) as follows:

Table 7-22 Typical IPv6 Extended Header Format (Traditional Representation)

1 0 1 2 3 4 5 6 7 8 9 0	2 0 1 2 3 4 5 6 7 8 9 0	3 0 1 2 3 4 5 6 7 8 9 0
Next Header Type	Length	

Table 7-23 lists the encoding of the *Next Header Type* field and information on determining each header type's length. Other IPv6 extension headers - not indicated in Table 7-23 - are not recognized by the 82599. Any processing of packet content that follows such extension headers is not supported.

Table 7-23 Header Type Encoding and Lengths

Header	Next Header Type	Header Length (units are bytes unless otherwise specified)
IPv6	6	Always 40 bytes
IPv4	4	Offset bits[7:4] Unit = 4 bytes
TCP	6	Offset Byte[12].Bits[7:4] Unit = 4 bytes
UDP	17	Always 8 bytes
Hop-by-Hop Options	0 - Note 1	8+Offset Byte[1]
Destination Options	60	8+Offset Byte (note 1)
Routing	43	8+Offset Byte (note 1)
Fragment	44	Always 8 bytes
Authentication	51	Note 3

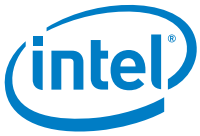


Table 7-23 Header Type Encoding and Lengths (Continued)

Header	Next Header Type	Header Length (units are bytes unless otherwise specified)
Encapsulating Security Payload	50	Note 3
No Next Header	59	Note 2

Notes:

1. Hop-by-hop options header is only found in the first next header type of an IPv6 header.
2. When no next header type is found, the rest of the packet should not be processed.
3. Encapsulated security payload packet handled by software — The 82599 cannot offload packets with this header type.

UDP/TCP Filter

This filter checks for a valid UDP or TCP header. The prototype next header values are 0x11 and 0x06, respectively.

7.1.12 SCTP Receive Offload

If a receive packet is identified as SCTP, the 82599 checks the CRC32 checksum of this packet and identifies this packet as SCTP. Software is notified of the CRC check via the L4I and L4E bits in the *Extended Status* field and *Extended Error* field in the Rx descriptor. The detection of an SCTP packet is indicated via the *SCTP* bit in the *Packet Type* field of the Rx descriptor. SCTP CRC uses the CRC32c polynomial as follows (0x11EDC6F41):

$$X_{32}+X_{28}+X_{27}+X_{26}+X_{25}+X_{23}+X_{22}+X_{20}+X_{19}+X_{18}+X_{14}+X_{13}+X_{11}+X_{10}+X_{9}+X_{8}+X_{6}+X_{0}$$

The checker assumes the following SCTP packet format.

Table 7-24 SCTP Header

0 1 2 3 4 5 6 7	1 8 9 0 1 2 3 4 5	2 6 7 8 9 0 1 2 3	3 4 5 6 7 8 9 0 1
Source Port		Destination Port	
Verification Tag			
CRC Checksum (CRC32c)			
Chunks 1..n			

7.1.13 Receive UDP Fragmentation Checksum

The 82599 might provide a receive fragmented UDP checksum offload for IPv4 non-tunneled packets. The RXCSUM.PCSD bit should be cleared and the RXCSUM.IPPCSE bit should be set to enable this mode.

The following table lists the outcome descriptor fields for the following incoming packets types.

Incoming Packet Type	Fragment Checksum	UDPV	UDPCS / L4CS
Non-IP packet	0	0	0
IPv6 Packet	0	0	Depends on transport UDP: 1 / 1 TCP: 0 / 1
Non fragmented IPv4 packet			
Fragmented IPv4 with protocol = UDP, first fragment (UDP protocol present)	The unadjusted 1's complement checksum of the IP payload	1 if the UDP header checksum is valid (not 0)	1 / 0
Fragmented IPv4, when not first fragment	The unadjusted 1's complement checksum of the IP payload	0	1 / 0

When the driver computes the 16-bit ones complement sum on the incoming packets of the UDP fragments, it should expect a value of 0xFFFF. Refer to Section 7.1.2.8.3 for supported packet formats.

7.2 Transmit Functionality

7.2.1 Packet Transmission

Transmit packets are made up of data buffers in host memory that are indicated to hardware by pointer and length pairs. These pointer and length pairs are named as transmit descriptors that are stored in host memory as well.

Software prepares memory structures for transmission by assembling a list of descriptors. It then indicates this list to hardware for updating the on-chip transmit tail pointer. Hardware transmits the packet only after it has completely fetched all packet data from host memory and deposited it into the on-chip transmit FIFO. This store and forward scheme enables hardware-based offloads such as TCP or UDP checksum computation, and many other ones detailed in this document while avoiding any potential PCIe under-runs.

7.2.1.1 Transmit Storage in System Memory

A packet (or multiple packets in transmit segmentation) can be composed of one or multiple buffers. Each buffer is indicated by a descriptor. Descriptors of a single packet are consecutive, while the first one points to the first buffer and the last one points to the last buffer (see Figure 7-14). The following rules must be kept:

- Address alignment of the data buffers can be on any byte boundary.
- Data buffers of any transmitted packet must include at least the 12 bytes of the source and destination Ethernet MAC addresses as well as the 2 bytes of the *Type/Len* field.
- A packet (or multiple packets in transmit segmentation) can span any number of buffers (and their descriptors) up to a limit of 40 minus WTHRESH minus 2 (see Section 7.2.3.3 for Tx Ring details and section Section 7.2.3.5.1 for WTHRESH details). For best performance it is recommended to minimize the number of buffers as possible.

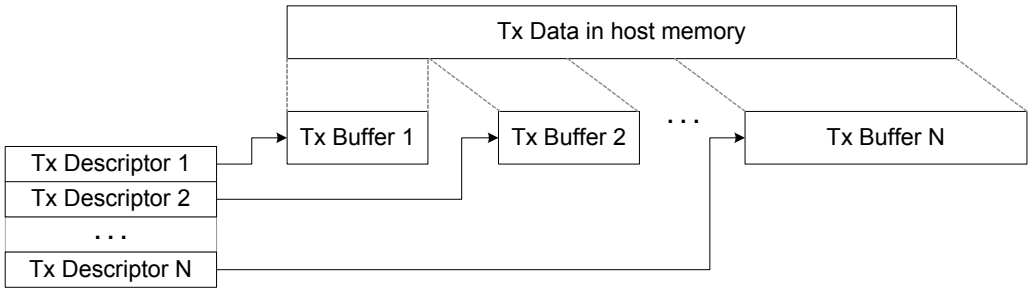


Figure 7-14 Tx Packet in Host Memory

7.2.1.2 Transmit Path in the 82599

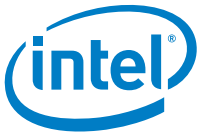
The transmit path in the 82599 consists of the following stages:

- Descriptor plane
  - The 82599 maintains a set of 128 on-die descriptor queues. Each queue is associated with a single descriptor ring in system memory. See Section 7.2.3.3 for more details on the Tx descriptor rings. Each on-die descriptor queue contains up to 40 descriptors.
  - A fetch mechanism loads Tx descriptors from the descriptor rings in system memory to the respective descriptor queues in the 82599. A descriptor fetch arbiter determines the order in which descriptors are fetched into the various on-die descriptor queues. See Section 7.2.3.4 for more details on the fetch mechanism.
  - An arbitration scheme determines the order in which descriptors are processed and requests are generated for data reads. These requests load packet data from system memory into a set of Tx packet buffers. The arbitration mechanism varies with configuration and is shown in Figure 7-17.
  - Once a packet has been fetched into a packet buffer, status is (optionally) written back into system memory. See Section 7.2.3.5 for more details.
- Packet plane (data plane)
  - Packet data is stored in up to eight packet buffers. The number and size of packet buffers vary with the mode of operation and is described in Section 7.2.1.2.2.
  - If more than a single packet buffer is enabled, an arbitration scheme determines the order in which packets are taken out of the packet buffers and sent to the MAC for transmission.The arbitration mechanism is shown in Figure 7-17.

7.2.1.2.1 Tx Queues Assignment

The 82599 supports a total of 128 queues per LAN port. Each Tx queue is associated with a packet buffer and the association varies with the operational mode. The following mechanisms impact the association of the Tx queues. These are described briefly in this section, and in full details in separate sections:

- Virtualization (VT) - In a virtualized environment, DMA resources are shared between more than one software entity (operating system and/or device driver). This is done through allocation of transmit descriptor queues to virtual partitions (VMM, IOVM, VMs, or VFs). Allocation of queues to virtual partitions is done in sets of queues of the same size, called queue pools, or **pools**. A pool is associated with a single virtual partition. Different queues in a pool can be associated with different packet buffers. For example, in a DCB system, each of the queues in a pool might belong to a different TC and therefore to a different packet buffer. The PFVFTE register contains a bit per VF. When the bit is set to 0b, packet transmission from the VF is disabled. When set to 1b, packet transmission from the VF is enabled.
- DCB — DCB provides QoS through priority queues, priority flow control, and congestion management. Queues are classified into one of several (up to eight) Traffic Classes (TCs). Each TC is associated with a single unique packet buffer.



- Transmit fanout — A single descriptor queue might be enough for a given functionality. For example, in a VT system, a single Tx queue can be allocated per VM. However, it is often the case that the data rate achieved through a single buffer is limited. This is especially true with 10 GbE, and traffic needs to be divided into several Tx queues in order to reach the desired data rate. Therefore, multiple queues might be provided for the same functionality.

Table 7-25 lists the queuing schemes. Selection of a scheme is done via the MTQC register.

Table 7-25 Tx Queuing Schemes

VT	DCB	Queues Allocation	Packet Buffers allocation
No	No	A single set of 64 queues is assigned to a single packet buffer. Queues 64…127 should not be used.	A single packet buffer for all traffic
No	Yes	Eight TCs mode – allocation of 32-32-16-16-8-8-8-8 queues for TC0-TC1-…- TC7, respectively. Four TCs mode — allocation of 64-32-16-16 queues for TC0-TC1-…- TC3, respectively.	A separate packet buffer is allocated to each TC (total of four or eight).
Yes	No	32 pools x 4 queues, or 64 pools x 2 queues	A single packet buffer for all traffic.
Yes	Yes	16 pools x 8 TCs, or 32 pools x 4 TCs	A separate packet buffer is allocated to each TC (total of four or eight).

Note: Software can use any number of queues per each TC or per each pool within the allocated ranges previously described by disabling any unused queue.

Programming MTQC must be done only during the init phase while software must also set RTTDCS.ARBDIS before configuring MTQC and then clear RTTDCS.ARBDIS afterwards.

Allocating descriptor queues to VFs uses a consistent indexing over the different Tx queuing schemes. The most significant bits of the queue index represent the VF index, and the least significant bits are either the TC index or are used by software to dispatch traffic according to a Transmit Side Scaling (TSS) algorithm — similar to RSS in the Rx path.

The Tx queue numbers associated with the TCs are listed in the following tables: Table 7-26 and Table 7-27.

Table 7-26 Tx Queues Indexing When VT-on

VT mode	Allocation of queue index bits according to						
	6	5	4	3	2	1	0
64 VFs + TSS	VF (63..0)						TSS
32 VFs + TSS *or* 4 TCs	VF (31 ..0)					TSS / TC	
16 VFs + 8 TCs	VF (15 ..0)				TC		

Table 7-27 Tx Queues Indexing When VT-off and DCB-on

TC mode	TCn	# of Qs	Queues attached to TCn
4 TCs	TC0	64	0xxxxxx
	TC1	32	10xxxxx
	TC2	16	110xxxx
	TC3	16	111xxxx
8 TCs	TC0	32	00xxxxx
	TC1	32	01xxxxx
	TC2	16	100xxxx
	TC3	16	101xxxx
	TC4	8	1100xxx
	TC5	8	1101xxx
	TC6	8	1110xxx
	TC7	8	1111xxx

Note: "x" refers to both 0 or 1, and is used by software to dispatch Tx flows via TSS algorithm.

7.2.1.2.2 Tx Packet Buffers

As previously described, the following modes exist for the 82599 packet buffers:

- A single 160 KB packet buffer that serves all Tx descriptor queues, leading to one single (or no) TC enabled, TC0
- Four 40 KB packet buffers, one per enabled TC, leading to four TCs, TC0 to TC3
- Eight 20 KB packet buffers, one per enabled TC, leading to eight TCs, TC0 to TC7

The size of the Tx packet buffer(s) is programmed via the TXPBSIZE registers, one register per TC. Null-sized packet buffer corresponds to a disabled TC.

Note: Setting the packet buffers' size leads to a different partition of a shared internal memory and must be done during boot, prior to communicating, and followed by a software reset.