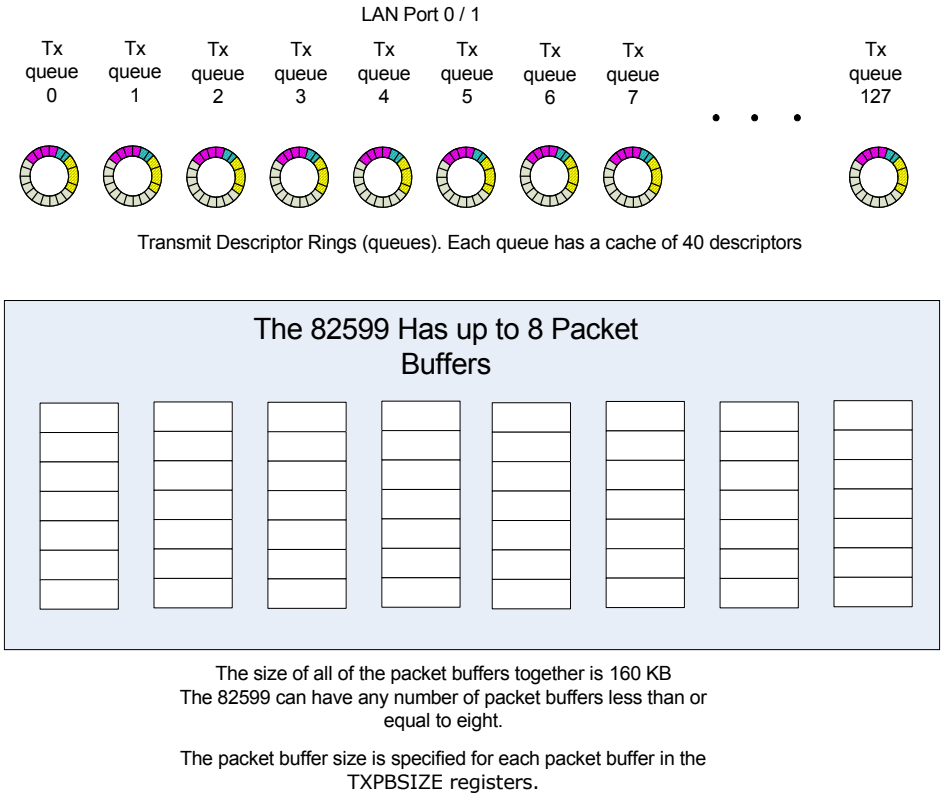


Figure 7-15 Tx Arbitration Schemes

7.2.1.2.3 Tx Arbitration Schemes

There are basically four Tx arbitration schemes, one per each combination of the DCB and Virtualization (VT) enabled/disabled modes. They are configured via the MTQC.MTQE register field.

DCB-on/VT-on

When both DCB and virtualization are enabled, queues are allocated to the packet buffers in a fixed manner, the same number of queues per each TC. Two DCB modes are supported, four TCs or eight TCs mode, according to coherent configuration made in registers TXPBSIZE and MTQC.

Descriptor Plane Arbiters and Schedulers:

- Transmit Rate-Scheduler — Once a frame has been fetched out from a transmit rate-limited queue, the next time another frame could be fetched from that queue is regulated by the transmit rate-scheduler. In the meantime, the queue is considered as if it was empty (such as switched-off) for the subsequent arbitration layers.

- VM Weighted Round Robin Arbiter — Descriptors are fetched out from queues attached to the same TC in a frame-by-frame weighted round-robin manner, while taking into account any rate limitation as previously described. Weights or credits allocated to each queue are configured via the RTTDT1C register. Bandwidth unused by one queue is reallocated to the other queues within the TC, proportionally to their relative bandwidth shares. TC bandwidth limitation is distributed across all the queues attached to the TC, proportionally to their relative bandwidth shares. Details on weighted round-robin arbiter between the queues can be found in Section 7.7.2.3. It is assumed traffic is dispatched across the queues attached to a same TC in a straightforward manner, according to the VF to which it belongs.
- TC Weighted Strict Priority Arbiter — Descriptors are fetched out from queues attached to different TCs in a frame-by-frame weighted strict-priority manner. Bandwidth unused by one TC is reallocated to the others, proportionally to their relative bandwidth shares. Link bandwidth limitation is distributed across all the TCs, proportionally to their relative bandwidth shares. Details on weighted strict-priority arbiter between the TCs can be found at Section 7.7.2.3. It is assumed (each) driver dispatches traffic across the TCs according to the 802.1p User Priority field inserted by the operating system and according to a user priority-to-TC Tx mapping table.

Packet Plane Arbiters:

- TC Weighted Strict Priority Arbiter — Packets are fetched out from the different packet buffers in a frame-by-frame weighted strict-priority manner. Weights or credits allocated to each TC (such as to each packet buffer) are configured via RTTPT2C registers, with the same allocation done at the descriptor plane. Bandwidth unused by one TC and link bandwidth limitation is distributed over the TCs as in the descriptor plane. Details on weighted strict-priority arbiter between the TCs can be found in Section 7.7.2.3.
- Priority Flow Control Packets are inserted with strict priority over any other packets.
- Manageability Packets are inserted with strict priority over data packets from the same TC, with respect to the bandwidth allocated to the concerned TC. TCs that belong to manageability packets are controlled by MNGTXMAP.MAP.

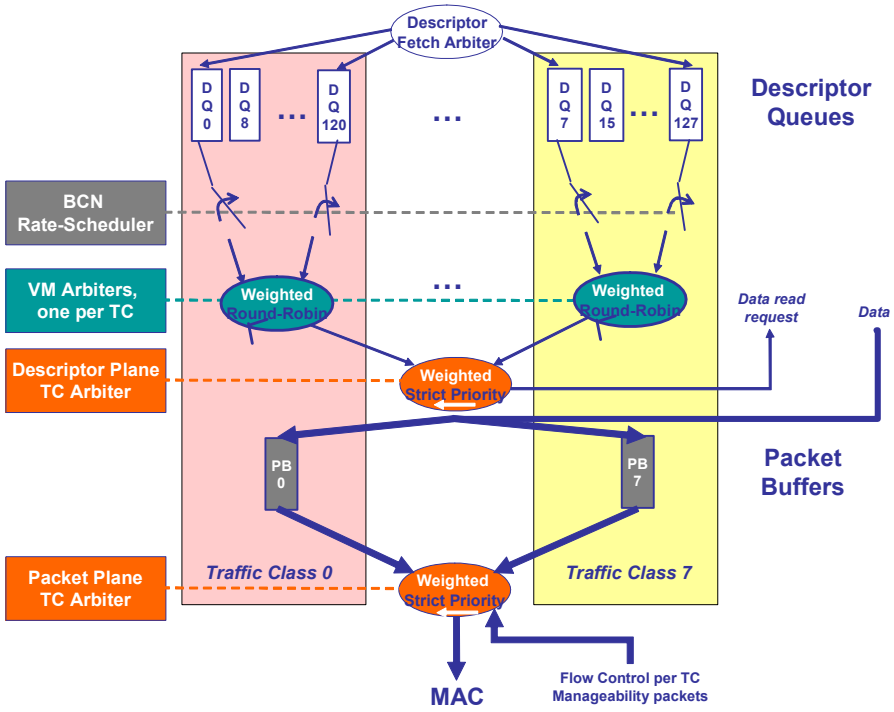


Figure 7-16 Transmit Architecture DCB-on/VT-on — Eight TCs Mode

Note: Replication of TC arbiters before and after the packet buffers is required to provide arbitration whether PCI bandwidth is smaller or greater than the link bandwidth, respectively.

DCB-on/VT-off

When DCB is enabled and virtualization disabled, queues are allocated to the packet buffers in a fixed manner according to the number of TCs. Two DCB modes are supported, four TCs or eight TCs mode, according to coherent configuration made in registers TXPBSIZE and MTQC. In Figure 7-17, eight TCs mode is shown.

- The unique difference with the DCB-on/VT-on arbitration scheme previously described is that the VM weighted round-robin arbiters are degenerated into simple frame-by-frame round-robin arbiters across the queues attached to the same TC. It is assumed driver dispatches traffic across the queues attached to a same TC according to hashing performed on MAC destination addresses. This is aimed to minimize crosstalk between transmit rate-limited and non-rate-limited flows.

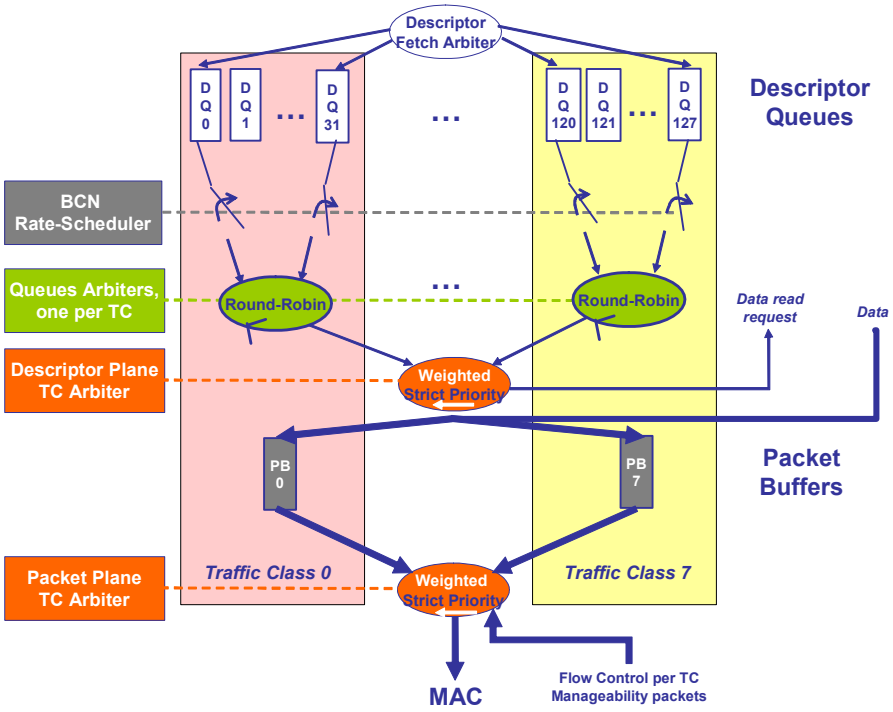


Figure 7-17 Transmit Architecture DCB-on/VT-off — Eight TCs Mode

DCB-off/VT-on

When DCB is disabled and virtualization enabled, all the 128 queues are allocated to a single packet buffer PB(0). Queues are grouped into 32 or 64 pools of 4 or 2 queues, respectively. The number of queue pools corresponds to the number of VFs exposed. Queues are attached to pools according to consecutive indexes

— For the 32 pools case, queues 0, 1, 2, 3 are attached to VF0, queues 4, 5, 6, 7 are attached to VF1, and so forth up to VF31.

— For the 64 pools case, queues 0 and 1 are attached to VF0, queues 2 and 3 are attached to VF1, and so forth up to VF63.

Descriptor Plane Arbiters:

- Descriptor Queues Round Robin Arbiter — Descriptors are fetched out from the internal descriptor queues attached to the same pool in a frame-by-frame round-robin manner. It is assumed driver dispatches traffic across the queues of a same pool according to some Transmit Side Scaling (TSS) algorithm similarly to what is done by hardware in the Rx path with RSS.
- VM Weighted Round Robin Arbiter — Descriptors are fetched out from queues attached to different pools in a frame-by-frame weighted round-robin manner. Weights or credits allocated to a pool are those allocated for the lowest queue of the pool via the RTTDT1C register. Bandwidth unused by one pool is reallocated to the others proportionally to their relative bandwidth shares. Link bandwidth limitation is distributed across all the pools, proportionally to their relative bandwidth shares.

Details on weighted round-robin arbiter between the pools can be found in Section 7.7.2.3.

Packet Plane Arbiter:

- Manageability packets are inserted with strict priority over data packets.

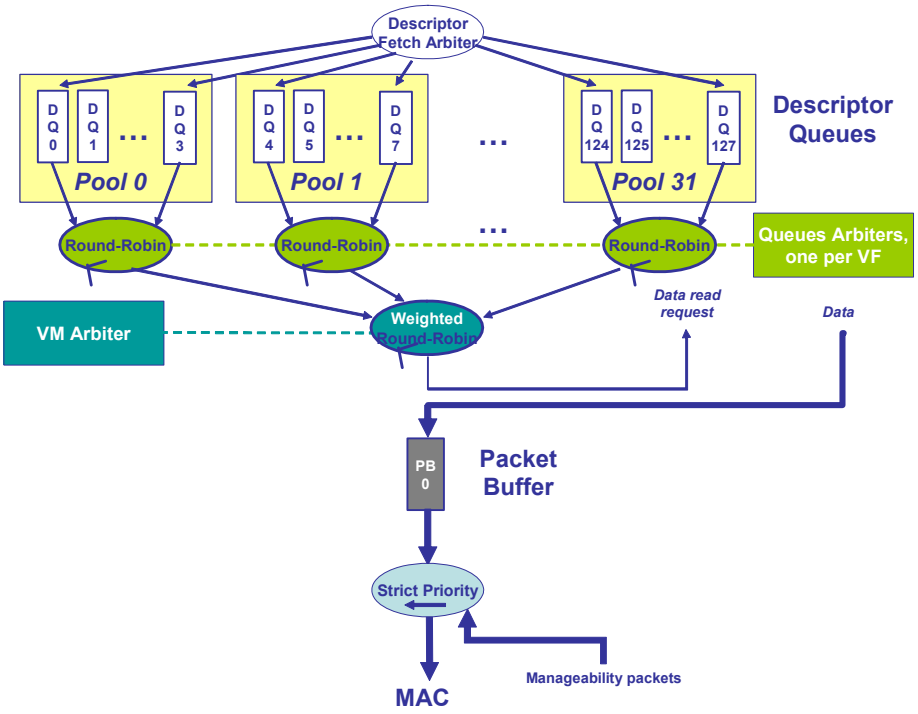


Figure 7-18 Transmit Architecture DCB-off/VT-on — 32 VFs

When both DCB and virtualization features are disabled, a single set of up to 64 queues is allocated to a single packet buffer PB(0).

Descriptor Plane Arbiter:

- Descriptor Queues Round Robin Arbiter — Descriptors are fetched out from the internal descriptor queues in a frame-by-frame round-robin manner. It is assumed driver dispatches traffic across the queues according to some TSS algorithm similarly to what is done by hardware in the Rx path with RSS.

Packet Plane Arbiter:

- Manageability packets are inserted with strict priority over data packets.

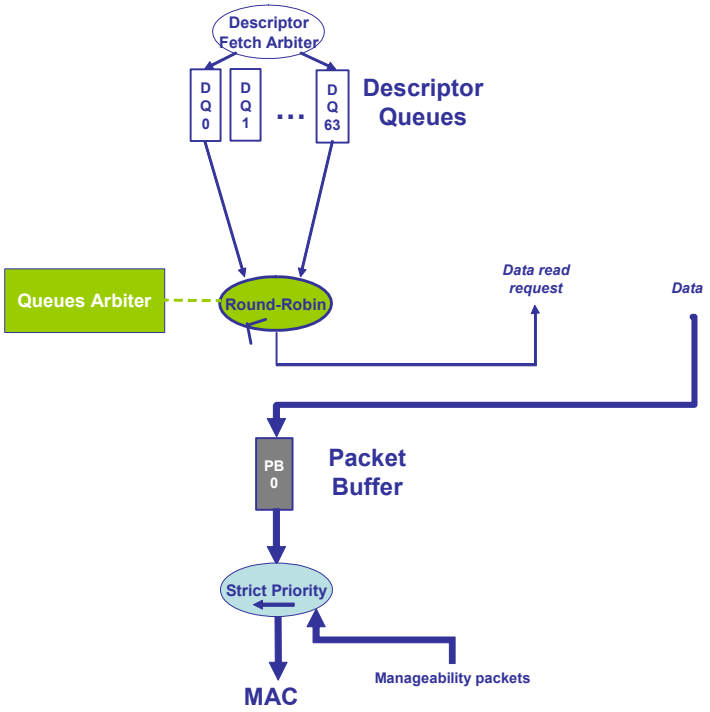


Figure 7-19 Transmit Architecture DCB-off/VT-off

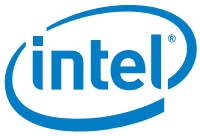
7.2.2 Transmit Contexts

The 82599 provides hardware checksum offload and TCP segmentation facilities. These features enable TCP and UDP packet types to be handled more efficiently by performing additional work in hardware, thus reducing the software overhead associated with preparing these packets for transmission. Part of the parameters used to control these features are handled through contexts.

A context refers to a set of parameters providing a particular offload functionality. These parameters are loaded by unique descriptors named transmit context descriptors. A transmit context descriptor is identified by the *DTYP* field (described later in this section) equals to 0x2.

The 82599 supports two contexts for each of its 128 transmit queues. The *IDX* bit contains an index to one of these two contexts. Each advanced data descriptor that uses any of the advanced offloading features must refer to a context by the *IDX* field.

Contexts can be initialized with a transmit context descriptor and then used for a series of related transmit data descriptors. Software can use these contexts as long lived ones, while one of the two contexts is used for checksum offload and the other one for transmit segmentation detailed in the following sections. The contexts should be modified when new offload parameters are required.



7.2.3 Transmit Descriptors

7.2.3.1 Introduction

The 82599 supports legacy descriptors and advanced descriptors.

Legacy descriptors are intended to support legacy drivers, in order to enable fast platform power up and to facilitate debug. The legacy descriptors are recognized as such based on *DEXT* bit (see the sections that follow). Legacy descriptors are not supported together with DCB, virtualization, Rate Scheduler, MACSec, and IPsec. These modes are recognized by a dedicated enable bit for each.

In addition, the 82599 supports two types of advanced transmit descriptors:

1. Advanced transmit context descriptor, DTYP = 0010b
2. Advanced transmit data descriptor, DTYP = 0011b

Note: DTYP = 0000b and 0001b are reserved values.

The transmit data descriptor (both legacy and advanced) points to a block of packet data to be transmitted. The advanced transmit context descriptor does not point to packet data. It contains control/context information that is loaded into on-chip registers that affect the processing of packets for transmission. The following sections describe the descriptor formats.

7.2.3.2 Transmit Descriptors Formats

7.2.3.2.1 Notations

This section defines the structure of descriptors that contain fields carried over the network. At the moment, the only relevant field is the *VLAN Tag* field.

The rule for VLAN tag is to use network ordering (also called big endian). It appears in the following manner in the descriptor:

Table 7-28 VLAN Tag

Byte address N + 1 -> first byte on the wire Bit 7 — first on the wire <- Bit 0			Byte address N -> second byte on the wire Bit 7 -> last on the wire — Bit 0
PRI (3 bits)	CFI	VID (4 bits)	VID (8 bits)

7.2.3.2.2 Legacy Transmit Descriptor Format

To select legacy mode operation, bit 29 (TDESC.DEXT) should be set to 0b. In this case, the descriptor format is defined as listed in Table 7-29. Address and length must be supplied by software on all descriptors. Bits in the command byte are optional, as are the *CSO*, and *CSS* fields.

Table 7-29 Transmit Descriptor (TDESC) Layout — Legacy Mode

	63 48	47 40	39 36	35 32	31 24	23 16	15 0
0	Buffer Address [63:0]						
8	VLAN	CSS	Rsvd	STA	CMD	CSO	Length

Table 7-30 Transmit Descriptor Write-Back Format — Legacy Mode

	63 48	47 40	39 36	35 32	31 24	23 16	15 0
0	Reserved				Reserved		
8	VLAN	CSS	Rsvd	STA	CMD	CSO	Length

Buffer Address (64) and Length (16)

The buffer address is a byte-aligned address. Length (TDESC.LENGTH) specifies the length in bytes to be fetched from the buffer address provided. The maximum length associated with a single descriptor is 15.5 KB while the total frame size must meet the maximum supported frame size. There is no limitation for the minimum buffer size.

Note: Descriptors with zero length (null descriptors) transfer no data. Null descriptors might appear only between packets and must have their *EOP* bits set.

Checksum Offset and Start — CSO (8) and CSS (8)

A *Checksum Offset* (TDESC.CSO) field indicates where, relative to the start of the packet, to insert a TCP checksum if this mode is enabled. A *Checksum Start* (TDESC.CSS) field indicates where to begin computing the checksum. Note that *CSO* and *CSS* are meaningful only in the first descriptor of a packet.

Both *CSO* and *CSS* are in units of bytes. These must both be in the range of data provided to the device in the descriptor. This means for short packets that are padded by software, *CSO* and *CSS* must be in the range of the unpadded data length, not the eventual padded length (64 bytes). The allowed ranges for *CSO* and *CSS* are:

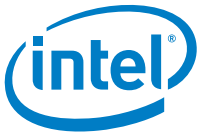
```
14 ≤ CSS ≤ unpadded packet length minus 1
CSS + 2 ≤ CSO ≤ unpadded packet length minus 4
```

For the 802.1Q header, the offset values depend on the VLAN insertion enable bit — the *VLE* bit. If they are not set (VLAN tagging included in the packet buffers), the offset values should include the VLAN tagging. If these bits are set (VLAN tagging is taken from the packet descriptor), the offset values should exclude the VLAN tagging.

Hardware does not add the 802.1q Ethertype or the VLAN field following the 802.1Q Ethertype to the checksum. So for VLAN packets, software can compute the values to back out only on the encapsulated packet rather than on the added fields.

Note: UDP checksum calculation is not supported by the legacy descriptor because the legacy descriptor does not support the translation of a checksum result of 0x0000 to 0xFFFF needed to differentiate between an UDP packet with a checksum of zero and an UDP packet without checksum.

Because the *CSO* field is eight bits wide, it puts a limit on the location of the checksum to 255 bytes from the beginning of the packet.



Note: CSO must be larger than CSS.

Software must compute an offsetting entry to back out the bytes of the header that should not be included in the TCP checksum and store it in the position where the hardware computed checksum is to be inserted.

Hardware adds the checksum at the byte offset indicated by the *CSO* field. Checksum calculations are for the entire packet starting at the byte indicated by the *CSS* field. The byte offset is counted from the first byte of the packet fetched from host memory.

Command Byte — CMD (8)

The CMD byte stores the applicable command and has the fields listed in Table 7-31.

Table 7-31 Transmit Command (TDESC.CMD) Layout

7	6	5	4	3	2	1	0
RSV	VLE	DEXT	RSV	RS	IC	IFCS	EOP

- RSV (bit 7) — Reserved
- VLE (bit 6) — VLAN Packet Enable

  When set to 1b, VLE indicates that the packet is a VLAN packet and hardware adds the VLAN header to the Tx packet. The VLAN Ethertype is taken from DMATXCTL.VT and the 802.1q VLAN tag is taken from the *VLAN* field in the Tx descriptor. See Section 7.4.5 for details about double VLAN.

Table 7-32 VLAN Tag Insertion Decision Table for VLAN Mode Enabled

VLE	Action
0	Send generic Ethernet packet.
1	Send 802.1Q packet; the *Ethernet Type* field comes from the *VET* field of the VLNCTRL register and the VLAN data comes from the *VLAN* field of the TX descriptor.

Note: This table is relevant only if VMVIR.VLANA = 00b (use descriptor command) for the queue.

- DEXT (bit 5) — Descriptor extension (zero for legacy mode)
- RSV (bit 4) — Reserved
- RS (bit 3) — Report Status - RS signals hardware to report the DMA completion status indication as well as triggering ITR. Hardware indicates a DMA completion by setting the *DD* bit in the Tx descriptor when TDWBAL[n].Head_WB_En = 0b or by Head Write-back if Head_WB_En = 1b (see Section 7.2.3.5.2). The *RS* bit is permitted only on descriptors that has the *EOP* bit set (last descriptor of a packet).

Note: Software should not set the *RS* bit when TXDCTL.WTHRESH is greater than zero. Instead, the hardware reports the DMA completion according to the WTHRESH rules (explained in Section 7.2.3.5.1). This note is relevant only for descriptor write back while in head write-back mode. WTRESH must also be set to zero.

When TXDCTL.WTHRESH = zero, software must set the *RS* bit on the last descriptor of every packet.

There are some exceptions for descriptor completion indication in head write-back mode. For more details see Section 7.2.3.5.2.

- IC (bit 2) — Insert Checksum - Hardware inserts a checksum at the offset indicated by the *CSO* field if the *Insert Checksum* bit (IC) is set.
- IFCS (bit 1) — Insert FCS - When set, hardware appends the MAC FCS at the end of the packet. When cleared, software should calculate the FCS for proper CRC check. There are several cases in which software must set IFCS as follows:
  - Transmitting a short packet while padding is enabled by the HLREG0.TXPADEN bit.
  - Checksum offload is enabled by the *IC* bit in the TDESC.CMD.
  - VLAN header insertion enabled by the *VLE* bit in the TDESC.CMD or by the PFVMVIR registers.
  - TSO or TCP/IP checksum offload using a context descriptor.
  - LinkSec offload is requested.

Note that TSO, Transmit Rate Scheduler, and LinkSec offload are relevant only to advanced Tx descriptors.

- EOP (bit 0) — End of Packet - A packet can be composed of multiple buffers (each of them indicated by its own descriptor). When EOP is set, it indicates the last descriptor making up the packet.

Note: *VLE*, *IFCS*, and *IC* fields should be set in the first descriptor of a packet. The *RS* bit can be set only on the last descriptor of a packet. The *DEXT* bit must be set to zero for all descriptors. The *EOF* bit is meaningful in all descriptors.

Transmitted.Status — STA (4)

DD (bit 0) — Descriptor Done Status

This bit provides a status indication that the DMA of the buffer has completed. Software might re-use descriptors with the *DD* bit set and any other descriptors processed by the hardware before this one. The other bits in the *STA* field are reserved.

Rsvd — Reserved (4)

VLAN (16)

The *VLAN* field is used to provide the 802.1q/802.1ac tagging information. The VLAN field is qualified on the first descriptor of each packet when the *VLE* bit is set to 1b. The *VLAN* field is provided in network order and is meaningful in the first descriptor of a packet. See Section 7.2.3.2.1 for more details.

Table 7-33 VLAN Field (TDESC.VLAN) Layout

15 13	12	11 0
PRI	CFI	VLAN

7.2.3.2.3 Advanced Transmit Context Descriptor

Table 7-34 Transmit Context Descriptor (TDESC) Layout — (Type = 0010)

	63	48	47	42	41	32	31	16	15	9	8	0
0	RSV		FCoEF		IPsec SA Index		VLAN		MACLEN		IPLEN/HEADLEN	
8	MSS		L4LEN		RSV	IDX	BCNTLEN	DEXT RSV	DTYP		TUCMD	IPsec ESP_LEN

	63	48	47	40	39	37	36	35	30	29	28 24	23	20	19	9	8	0

IPLEN/HEADLEN (9)

- IPLEN — for IP packets:

  This field holds the value of the IP header length for the IP checksum offload feature. If an offload is requested, IPLEN must be greater than or equal to 20, and less than or equal to 511. For IP tunnel packets (IPv4-IPv6) IPLEN must be defined as the length of the two IP headers. The hardware is able to offload the L4 checksum calculation while software should provide the IPv4 checksum. For IPsec packets, it is the sum of IP header length plus IPsec header length.

- HEADLEN — for FCoE packets:

  This field indicates the size (in bytes) of the FCoE frame header. The frame header includes the MAC header, optional VLAN and FCoE header(s) as shown in Figure 7-47. HEADLEN does not include the LinkSec header if it exists. HEADLEN is meaningful only if transmit FCoE offload is enabled by setting the *FCoE* bit in the TUCMD field. HEADLEN that matches Figure 7-47 equals 56 or 64 for packets without FC extended headers or packets with VFT header respectively. The 82599 supports FC extended headers only for single send. Segmentation offload can be used only when extended headers are not present.

MACLEN (7)

- For nonFCoE packets:

  This field indicates the length of the MAC header. When an offload is requested, the TSE bit (in the advanced transmit data descriptor) or *IXSM* bit or *TXSM* bit are set, MACLEN must be larger than or equal to 14, and less than or equal to 127. This field should include only the part of the L2 header supplied by the driver and not the parts added by hardware. Table 7-35 lists the value of MACLEN in the different cases.

Table 7-35 MACLEN Values

SNAP	Regular VLAN	Extended VLAN	MACLEN
No	By hardware or no	No	14
No	By hardware or no	Yes	18
No	By software	No	18
No	By software	Yes	22
Yes	By hardware or no	No	22

Table 7-35 MACLEN Values (Continued)

SNAP	Regular VLAN	Extended VLAN	MACLEN
Yes	By hardware or no	Yes	26
Yes	By software	No	26
Yes	By software	Yes	30

- For FCoE packets:

  This field is a byte offset to the last Dword of the FCoE header (supplied by the driver) that includes the SOF flag. The FC frame header starts four bytes after the MACLEN as shown in Figure 7-47. The MACLEN that matches Figure 7-47 equals 28.

VLAN (16)

This field contains the 802.1Q VLAN tag to be inserted in the packet during transmission. This VLAN tag is inserted when a packet using this context has its DCMD.VLE bit is set. This field should include the entire 16-bit VLAN field including CFI and priority fields as listed in Table 7-33.

Note that the *VLAN* field is provided in network order. See Section 7.2.3.2.1.

Ipsec SA IDX (10) – IPsec SA Index. If an IPsec offload is requested for the packet (IPSEC bit is set in the advanced Tx data descriptor), indicates the index in the SA table where the IPsec key and SALT are stored for that flow.

FCoEF (6) — see the following:

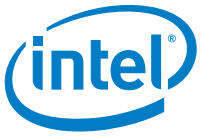
- EOF (bits 1:0) — End of frame delimiter index.
- ORIE (bit 4) — Orientation relative to the last frame in an FC sequence.

The *EOF* and *ORIE* fields define the *EOF* that is inserted by hardware. In a single packet send, the *EOF* field is defined completely by the *EOF* setting while in TSO mode, the *EOF* field is defined by the *EOF* and the *ORIE* bits as listed in Table 7-36. The values EOF0...EOF3 are taken from the TEOFF register.

Table 7-36 EOF Codes in TSO

EOF bits (1:0)	ORIE bit (4)	E-EOF code in a single packet send	FCoE Large send		
			E-EOF code in the last frame	E-EOF code in other frames	TSO that ends up in a single packet
00 (EOFn)	0 (not a sequence end)	EOF0 (EOFn)	EOF0 (EOFn)	EOF0 (EOFn)	EOF0 (EOFn)
00 (EOFn)	1 (sequence end)	EOF0 (EOFn)	EOF1 (EOFt)	EOF0 (EOFn)	EOF1 (EOFt)
01 (EOFt)	1 (don't care)	EOF1 (EOFt)	n/a	n/a	EOF1 (EOFt)
10 (EOFni)	1 (don't care)	EOF2 (EOFni)	n/a	n/a	EOF2 (EOFni)
11 (EOFa)	1 (don't care)	EOF3 (EOFa)	n/a	n/a	EOF3 (EOFa)

- SOF (bit 2) — Start of frame delimiter index.
- ORIS (bit 5) — Orientation relative to the first frame in an FC sequence.



In a single packet send, SOF is taken from the data buffer. In TSO, hardware places the SOF in the transmitted packet replacing the data buffer content. The *SOF* and *ORIS* bits in the context descriptor define the SOF that is placed by the hardware as listed in Table 7-37. The values SOF0...SOF3 are taken from the SOFF register.

Table 7-37 SOF Codes

SOF bit (2)	ORIS bit (5)	SOF code in the first frame	SOF code in other frames	SOF code in a single packet
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 start)	SOF2 (SOFn2)	SOF2 (SOFn2)	SOF2 (SOFn2)

- PARINC (bit 3) — When this bit is set, hardware relates to the *PARAM* field in the FC header as relative offset. In this case, hardware increments the *PARAM* field in TSO by an MSS value on each transmitted packet of the TSO. Software should set the *PARINC* bit when it sets the *Relative Offset Present* bit in the F_CTL.

RSV(16)

Reserved

IPS_ESP_LEN(9) - Size of the ESP trailer and ESP ICV appended by software. Meaningful only if the IPSEC_TYPE bit is set in the TUCMD field and to single send packets for which the IPSEC bit is set in their advanced Tx data descriptor.

TUCMD (11)

- RSV (bit 10-7) — Reserved
- FCoE (bit 6) — This bit defines the context descriptor and the associated data descriptors as FCoE frame type. See Section 7.13.2 for a description of the offload provided by the hardware while transmitting a single frame and TSO.
- Encryption (bit 5) — ESP encryption offload is required. Meaningful only to packets for which the IPSEC bit is set in their advanced Tx data descriptor.
- IPSEC_TYPE (bit 4) — Set for ESP. Cleared for AH. Meaningful only to packets for which the IPSEC bit is set in their advanced Tx data descriptor.
- L4T (bit 3:2) — L4 Packet TYPE (00: UDP; 01: TCP; 10: SCTP; 11: RSV)
- IPV4(bit 1) — IP Packet Type: When 1b, IPv4; when 0b, IPv6
- SNAP (bit 0) — SNAP indication

DTYP (4)

This field is always 0010b for this type of descriptor.

RSV(1)

Reserved

DEXT (1) — Descriptor extension (one for advanced mode)

BCNTLEN(6) — For rate limited queues this field must be set to 0x3F.

IDX (1)

The context descriptor is posted to a context table in hardware. There are two context tables per queue. The IDX is the index of the context tables.

Note: Because the 82599 supports only two context descriptors per queue, the two MS bits are reserved and should be set to 0b.

RSV(1)

L4LEN(8)

This field holds the layer 4 header length. If TSE is set, this field must be greater than or equal to 8 and less than or equal to 255. Otherwise, this field is ignored. Note that for UDP segmentation the L4 header size equals 8 and for TCP segmentation (with no TCP options) it equals 20.

MSS (16)

This field controls the Maximum Segment Size. This specifies the maximum protocol payload segment sent per frame, not including any header. MSS is ignored when DCMD.TSE is not set.

TCP / UDP Segmentation

The total length of each frame (or segment) excluding Ethernet CRC as follows. Note that the last packet of a TCP segmentation might be shorter.

```
MACLEN + 4(if VLE set)  + IPLEN + L4LEN + MSS + [PADLEN + 18] (if ESP
packet)
```

PADLEN ranges from zero to three in Tx and is the content of the ESP Padding Length field that is computed when offloading ESP in cipher blocks of 16 bytes (AES-128) with respect to the following alignment formula:

```
[L4LEN + MSS + PADLEN + 2] modulo(4) = 0
```

For a single send the IPS_ESP_LEN equals to PADLEN + 18.

Note: The headers lengths must meet the following: MACLEN + IPLEN + L4LEN <= 512

FCoE Segmentation

The total length of each frame (or segment) excluding Ethernet CRC equals to:

```
MACLEN + 4(if VLE set)  + 8 (FC CRC + EOF)
```

Note: For FCoE packets, the maximum segment size defines the FC payload size in all packets but the last one, which can be smaller.

The context descriptor requires valid data only in the fields used by the specific offload options. The following table lists the required valid fields according to the different offload options.

Table 7-38 Valid Fields by Offload Option

	Context Fields ->	FCoE	FCoEF	VLAN	MACLEN	IPLEN/HEADLEN	L4LEN	SNAP	IPV4	L4T	Encryption	IPSECTYPE	SAIDX	ESP_LEN	MSS	BCNTLEN	CC (data descriptor)
Required Offload	VLAN insertion			yes												yes	
	IPv4 XSUM	n/a	n/a		yes	yes			1							yes	0
	L4 XSUM	n/a	n/a		yes	yes				yes						yes	0
	TCP/UDP Seg	n/a	n/a		yes	yes	yes	yes	yes	yes					yes	yes	0
	FCoE CRC	yes	yes		yes	yes	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a		yes	1
	FCoE Seg	yes	yes		yes	yes	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	yes	yes	1
	IPSec ESP	n/a	n/a		yes	yes			yes		yes	yes	yes	yes		yes	1
	IPSec AH	n/a	n/a		yes	yes			yes		yes	yes	yes	n/a		yes	1
	Tx switch	n/a	n/a		yes	yes	yes		yes	yes	n/a	n/a	n/a	n/a		yes	1

Note: All fields that are not used in the context descriptor must be set to zero.

7.2.3.2.4 Advanced Transmit Data Descriptor

Table 7-39 Advanced Transmit Data Descriptor Read Format

0	Address[63:0]									
8	PAYLEN	POPTS	CC	IDX	STA	DCMD	DTYP	MAC	RSV	DTALEN
	63 46	45 40	39	38 36	35 32	31 24	23 20	19 18	17 16	15 0

Table 7-40 Advanced Transmit Data Descriptor Write-back Format

0	RSV		
8	RSV	STA	RSV
	63 36	35 32	31 0

General Rule for all Fields

When a packet spreads over multiple descriptors, all of the descriptor fields are valid only on the first descriptor of the packet, except for *RS* and *EOP* bits, which are set on the last descriptor of the packet.

Address (64)

This field holds the physical address of a data buffer in host memory, which contains a portion of a transmit packet. This field is meaningful in all descriptors.

DTALEN (16)

This field holds the length in bytes of data buffer at the address pointed to by this specific descriptor. This field is meaningful in all descriptors. The maximum length is 15.5 KB with no limitations on the minimum size. Refer to the comment on descriptors with zero length described in the sections that follow.

RSV(2)

Reserved

MAC (2)

See the following. This field is meaningful on the first descriptor of the packet(s).

- ILSec (bit 0) — Apply LinkSec on packet. When set, hardware includes the LinkSec header (SecTAG) and LinkSec header digest (signature). The LinkSec processing is defined by the *Enable Tx LinkSec* field in the LSECTXCTRL register. The *ILSec* bit in the packet descriptor should not be set if LinkSec processing is not enabled by the *Enable Tx LinkSec* field. If the *ILSec* bit is set erroneously while the *Enable Tx LinkSec* field is set to 00b, then the packet is dropped.
- 1588 (bit 1) — IEEE1588 time stamp packet.

DTYP (4)

0011b for advanced data descriptor. DTYP should be valid in all descriptors of the packet(s).

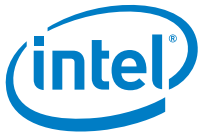
DCMD (8)

See the following:

- TSE (bit 7) — Transmit Segmentation Enable: This bit indicates a TCP or FCoE segmentation request. When *TSE* is set in the first descriptor of a TCP or FCoE packet, hardware must use the corresponding context descriptor in order to perform segmentation.

Note: It is recommended that HLREG0.TXPADEN be enabled when TSE is used since the last frame can be shorter than 60 bytes — resulting in a bad frame.

- VLE (bit 6) — VLAN Packet Enable: This bit indicates that the packet is a VLAN packet (hardware must add the VLAN Ethertype and an 802.1q VLAN tag to the packet).
- DEXT (bit 5) — Descriptor Extension: This bit must be one to indicate advanced descriptor format (as opposed to legacy).
- Rsv (bit 4) — Reserved
- RS (bit 3) — Report Status: See the description in the legacy transmit descriptor in Section 7.2.3.2.2.
- Rsv (bit 2) — Reserved
- IFCS (bit 1) — Insert FCS: When this bit is set, the hardware appends the MAC FCS at the end of the packet. When cleared, software should calculate the FCS for proper CRC check. There are several cases in which software must set IFCS as follows:
  - — Transmitting a short packet while padding is enabled by the HLREG0.TXPADEN bit.
  - — Checksum offload is enabled by the either *IC, TXSM* or *IXSM* bits in the TDESC.DCMD.



— VLAN header insertion enabled by the *VLE* bit in the TDESC.DCMD.

— FC CRC (FCoE) offload is enabled by the *FCoE* bit in the transmit context descriptor.

— TCP or FCoE segmentation offload enabled by the *TSE* bit in the TDESC.DCMD.

- EOP (bit 0) — End of Packet: A packet might be composed of multiple buffers (each of them is indicated by its own descriptor). When EOP is set, it indicates the last descriptor making up the packet. In transmit segmentation (explained later on in this section) the EOP flag indicates the last descriptor of the last packet of the segmented transmission.

Note: *TSE*, *VLE* and *IFCS* fields should be set in the first descriptor of the packet(s). The *RS* bit can be set only on the last descriptor of the packet. The *EOP* bit is valid in all descriptors. The *DEXT* bit must be set to 1b for all descriptors.

Descriptors with zero length, transfer no data. If the *RS* bit in the command byte is set, then the *DD* field in the status word is not written when hardware processes them.

STA (4)

- Rsv (bit 3:1) — Reserved
- DD (bit 0) — Descriptor Done: The *DD* bit provides a status indication that the DMA of the buffer has completed. Software might re-use descriptors with the *DD* bit set, and any other descriptors processed by hardware before this one. In TSO, the buffers that include the TSO header are used multiple times during transmission and special considerations should be made as described in Section 7.2.4.2.2.

IDX (3)

This field holds the index into the hardware context table to indicate which of the two per-queue contexts should be used for this request. If no offload is required and the *CC* bit is cleared, this field is not relevant and no context needs to be initiated before the packet is sent. See Table 7-38 for details of which packets requires a context reference. This field is relevant only on the first descriptor of the packet(s).

CC (1)

Check Context bit — When set, a Tx context descriptor indicated by IDX index should be used for this packet(s). The *CC* bit should be set in the following cases:

1. Non-zero *BCNTLEN* field is required (defined in the context descriptor).
2. Any FCoE offload is required.
3. Tx switching is enabled.

POPTS (6)

This field is relevant only on the first descriptor of the packet(s).

- Rsv (bits 5:3) — Reserved
- IPSEC (bit 2) — Ipsec offload request
- TXSM (bit 1) — Insert TCP/UDP Checksum: When set to 1b, the L4 checksum must be inserted. In this case, TUCMD.LP4 indicates whether the checksum is TCP or UDP or SCTP. When DCMD.TSE is set, TXSM must be set to as well. If this bit is set, the packet should at least contain an L4 header.

- IXSM (bit 0) — Insert IP Checksum: This field indicates that IP checksum must be inserted. In IPv6 mode, it must be reset to 0b. If DCMD.TSE and TUCMD.IPV4 are set, IXSM must be set as well. If this bit is set, the packet should at least contain an IP header.

PAYLEN (18)

PAYLEN indicates the size (in byte units) of the data buffer(s) in host memory for transmission. In a single-send packet, PAYLEN defines the entire packet size fetched from host memory. It does not include the fields that hardware adds such as: optional VLAN tagging, the FCoE trailer containing the FC CRC and EOF (for FCoE packets), Ethernet CRC or Ethernet padding.

When LinkSec offload is enabled, the PAYLEN field does not include the LinkSec encapsulation. When IPsec offload is enabled, the PAYLEN field does not include the ESP trailer added by hardware. In TSO (regardless if it is transmitted on a single or multiple packets), the PAYLEN defines the protocol payload size fetched from host memory. In TCP or UDP segmentation offload, PAYLEN defines the TCP/UDP payload size. In FCoE TSO offload, the PAYLEN field defines the FC payload size. It includes the FC option headers (if present) and the FC data payload but excludes the FCoE trailer containing the FC CRC and EOF.

This field is relevant only on the first descriptor of the packet(s). The minimum transmitted packet size excluding VLAN padding and CRC bytes is 17 and the PAYLEN size should meet this limitation. On a single-packet send, the maximum size of the PAYLEN is dictated by the maximum allowed packet size which is 15.5 KB. On TSO, the maximum PAYLEN can be up to 2^{18}-1.

7.2.3.3 Transmit Descriptor Ring

The transmit descriptor ring structure (shown in Figure 7-20) uses a contiguous memory space. A set of four registers (described later in this section) maintain the transmit descriptor ring in the host memory. Hardware maintains internal circular queues of 40 descriptors per queue to hold the descriptors that were fetched from the software ring.

Descriptors handled to hardware should not be manipulated by software until hardware completes its processing. It is indicated by advancing the head pointer beyond these descriptors.

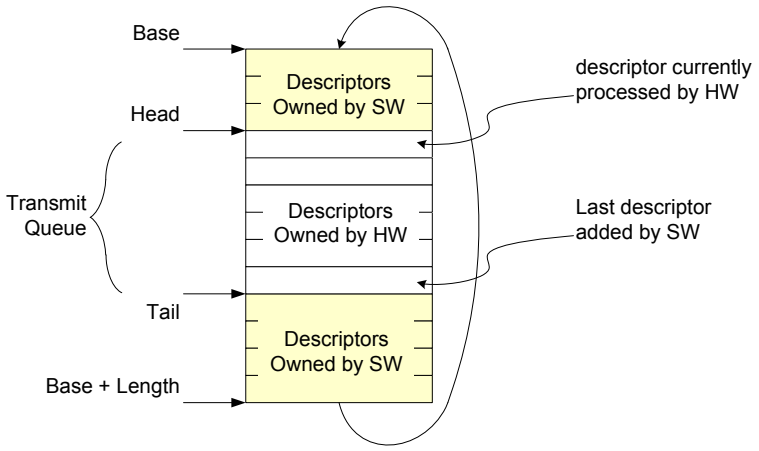


Figure 7-20 Transmit Descriptor Ring Structure

The transmit descriptor ring is defined by the following registers:

- Transmit Descriptor Base Address register (TDBA 0-127) — This register indicates the start address of the descriptor ring buffer in the host memory; this 64-bit address is aligned on a 16-byte boundary and is stored in two consecutive 32-bit registers. Hardware ignores the lower four bits.
- Transmit Descriptor Length register (TDLEN 0-127) — This register determines the number of bytes allocated to the circular buffer. This value must be 0 modulo 128.
- Transmit Descriptor Head register (TDH 0-127) — This register holds a value that is an offset from the base and indicates the in-progress descriptor. There can be up to 64 K minus 8 descriptors in the circular buffer. The transmit queue consists of the descriptors between the head and tail pointers. Transmission starts with the descriptor pointer by the head registers. When the DMA engine processes a descriptor, it might optionally write back the completed descriptor and then advance the head pointer. It then processes the next descriptor up to the point that the head pointer reaches the tail. Head equals tail means that the transmit queue in host memory is empty. Reading this register indicates the hardware progress to the software. All descriptors behind the head pointer and in front of tail register are owned by the software. The other descriptors are owned by the hardware and should not be modified by the software.
- Transmit Descriptor Tail register (TDT 0-127) — This register holds a value, which is an offset from the base, and indicates the location beyond the last descriptor hardware can process. Software adds new descriptors to the ring by writing descriptors in the circular buffer pointed by the tail pointer. The new descriptor(s) are indicated to hardware by updating the tail pointer one descriptor above the last added descriptor. Note that a single packet or TSO might be composed of multiple descriptors. The transmit tail pointer should never point to the middle of a packet or TSO, which might cause undesired software/hardware races.

Software might detect which packets have already been processed by hardware using the following:

- Read the TDH head register to determine which packets (those logically before the head) have been transferred to the on-chip FIFO or transmitted. This method is not recommended as races between the internal update of the head register and the actual write back of descriptors can occur.
- When head write back is enabled (TDWBAL[n].Head_WB_En = 1b) software might read the image of the head pointer in host memory at the address defined by TDWBAH[n]/TDWBAL[n] pair. Hardware updates the head image in host memory by completed descriptors as described in Section 7.2.3.5.2.
- When head write back is not enabled (TDWBAL[n].Head_WB_En = 0b), software might track the *DD* bits in the descriptor ring. Descriptor write back is controlled by the *RS* bit and the WTHRESH setting as well as interrupt assertion.
- Issue an interrupt. An interrupt condition is generated each time a packet was transmitted or received and a descriptor was write back or transmit queue goes empty (EICR.RTxQ[0-19]). This interrupt can either be enabled or masked.

All of the registers controlling the descriptor rings behavior should be set before transmit is enabled.

7.2.3.4 Transmit Descriptor Fetching

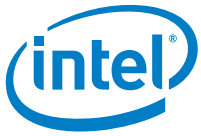
The 82599 fetches new descriptors as required for packet transmission depending on its on-die descriptor buffer state:

Fetch — The on-chip descriptor buffer is empty or contains less descriptors than a complete packet.

- A fetch starts as soon as any descriptors are made available (host writes to the tail pointer).
- A request is issued for any available descriptors up to the size of the on-die buffer.
- Once the sum of on-die descriptors and requested descriptors is more than required for a single packet, the buffer transitions to the pre-fetch state.
- If several on-chip descriptor queues are empty simultaneously, queues are served in round robin arbitration except the queues indicated as strict priority which are served first.

Pre-Fetch — The on-chip descriptor buffer becomes almost empty while there are enough descriptors in the host memory.

- The on-chip descriptor buffer is defined as almost empty if it contains less descriptors then the threshold defined by TXDCTL[n].PTHRESH
- The transmit descriptor contains enough descriptors if it includes more ready descriptors than the threshold defined by TXDCTL[n].HTHRESH
- In pre-fetch mode descriptors are fetched only after there are no other DMA activity of greater priority as: transmit descriptor fetch; status write-backs or packet data transfers)
- A request is issued for any available descriptors up to the capacity of the on-die buffer.



- If several on-chip descriptor queues are in this situation simultaneously, queues are served in round robin arbitration except the queues indicated as strict priority which are served first.

Idle — Requests are not issued. This is the state reached when none of the previous states apply.

Note: Software must update the Tail register on packet boundaries. That is, the last valid descriptor might not be a context descriptor and must have the *EOP* bit set.

7.2.3.4.1 Transmit Descriptor Fetch and Write-back Settings

This section describes the settings of transmit descriptor thresholds. It relates to fetch thresholds described above as well as the write-back threshold (WTHRESH) when operating in descriptor write-back mode which is described in Section 7.2.3.5.1.

- Transmit descriptor fetch setting is programmed in the TXDCTL[n] register per queue. The default settings of PTHRESH, HTHRESH and WTHRESH are zero’s.
- In order to reduce transmission latency, it is recommended to set the PTHRESH value as high as possible while the HTHRESH and WTHRESH as low as possible (down to zero).
- In order to minimize PCIe overhead the PTHRESH should be set as low as possible while HTHRESH and WTHRESH should be set as high as possible.
- The sum of PTHRESH plus WTHRESH must not be greater than the on-chip descriptor buffer size
- Some practical rules
  - CPU cache line optimization: Assume ‘N’ equals the CPU cache line divided by 16 (descriptor size). Then, in order to align descriptors pre-fetch to CPU cache line (in most cases), it is advised to set PTHRESH to the on-chip descriptor buffer size minus ‘N’ and HTHRESH to ‘N’. In order to align descriptor write back to the CPU cache line it is advised to set WTHRESH to either 'N' or even 2 times 'N'. Note that partial cache line writes might significantly degrade performance. Therefore, it is highly recommended to follow this advice.
  - Minimizing PCIe overhead: As an example, setting PTHRESH to the on-chip descriptor buffer size minus 16 and HTHRESH to 16 minimizes the PCIe request and header overhead to ~20% of the bandwidth required for the descriptor fetch.
  - Minimizing transmission latency from tail update: Setting PTHRESH to the on-chip descriptor buffer size minus ‘N’ (‘N’ previously defined) while HTHRESH and WTHRESH to zero.
  - Threshold settings in DCB mode: Note that only values of PTHRESH equals on-chip descriptor buffer size minus 8 and HTHRESH equals 4 were thoroughly tested.

Note: As previously described, device setting is a trade off between overhead (translated to performance) and latencies. It is expected that some level of optimization is done at software driver development phase. Customers who want better performance might need to adjust the threshold values according to the previous guidelines while optimizing to specific platform and targets.

7.2.3.5 Transmit Write Back

The 82599 periodically updates software on its progress in processing transmit buffers. Two methods are described for doing so:

- Updating by writing back into the Tx descriptor
- Update by writing to the head pointer in system memory

7.2.3.5.1 Tx Descriptor Write Back

When the TXDCTL[n].WTHRESH equals zero, descriptors are written back for those descriptors with the *RS* bit set. When the TXDCTL[n].WTHRESH value is greater than zero, descriptors are accumulated until the number of accumulated descriptors equals the TXDCTL[n].WTHRESH value, then these descriptors are written back. Accumulated descriptor write back enables better use of the PCIe bus and memory bandwidth.

Any descriptor write back includes the full 16 bytes of the descriptor.

Descriptors are written back in one of three cases:

- TXDCTL[n].WTHRESH = 0 and a descriptor that has *RS* set is ready to be written back.
- TXDCTL[n].WTHRESH > 0 and TXDCTL[n].WTHRESH descriptors have accumulated.
- TXDCTL[n].WTHRESH > 0 and the corresponding EITR counter has reached zero. The timer expiration flushes any accumulated descriptors and sets an interrupt event (TXDW).

An additional mode in which transmit descriptors are not written back at all and the head pointer of the descriptor ring is written instead is described in the following section.

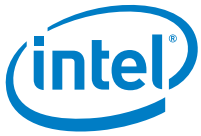
7.2.3.5.2 Tx Head Pointer Write Back

In legacy hardware, transmit requests are completed by writing the *DD* bit to the transmit descriptor ring. This causes cache thrash since both the driver and hardware are writing to the descriptor ring in host memory. Instead of writing the *DD* bits to signal that a transmit request is complete, hardware can write the contents of the descriptor queue head to host memory. The driver reads that memory location to determine which transmit requests are complete. In order to improve the performance of this feature, the driver needs to program DCA registers to configure which CPU will be processing each TX queue.

The head pointer is reflected in a memory location that is allocated by software for each queue.

Rules for head pointer write back:

- Head write back occurs if TDWBAL[n].Head_WB_En is set for this queue, and the *RS* bit is set in the Tx descriptor, following its corresponding data upload into packet buffer.
  - If the head write-back feature is enabled, software must set WTHRESH to 0x0 while only descriptors with the RS *bit* set, generate header write back.



— Note that the head pointer write back does not hold transmission. Instead, if packets with the *RS* bit are transmitted fast enough, it might happen that the header pointer write back is not updated for each and every packet. In addition, it might happen that the head pointer write back might be updated up to descriptors that do not have the *RS* bit set. In such cases, hardware might report a completion of a descriptor that might not be the last descriptor in a TSO or even the last descriptor in a single packet.

The driver has control of this feature per queue through the TDWBAL and TDWBAH registers.

The low register's LSB hold the control bits.

- The Head_WB_EN bit enables activation of tail write back. In this case, no descriptor write back is executed.
- The 30 upper bits of this register hold the lowest 32 bits of the head write-back address, assuming that the two last bits are zero.

The high register holds the high part of the 64-bit address.

Note: Hardware writes a full Dword when writing this value, so software should reserve enough space for each head value and make sure the TDBAL value is Dword-aligned.

7.2.4 TCP and UDP Segmentation

Hardware TCP segmentation is one of the offloading options supported by the Windows* and Linux* TCP/IP stack. This is often referred to as Large Send offloading or TSO. This feature enables the TCP/IP stack to pass to the network device driver a message to be transmitted that is bigger than the Maximum Transmission Unit (MTU) of the medium. It is then the responsibility of the device driver and hardware to divide the TCP message into MTU size frames that have appropriate layer 2 (Ethernet), 3 (IP), and 4 (TCP) headers. These headers must include sequence number, checksum fields, options and flag values as required. Note that some of these values (such as the checksum values) are unique for each packet of the TCP message, and other fields such as the source IP address is constant for all packets associated with the TCP message.

Similar to TCP segmentation, the 82599 also provides a capability to offload UDP segmentation. Note that current UDP segmentation offload is not supported by any standard OS.

Note: CRC appending (HLREG0.TXCRCEN) must be enabled in TCP / UDP segmentation mode because CRC is inserted by hardware.

Padding (HLREG0.TXPADEN) must be enabled in TCP / UDP segmentation mode, since the last frame might be shorter than 60 bytes — resulting in a bad frame if TXPADEN is disabled.

The offloading of these mechanisms to the device driver and the 82599 saves significant CPU cycles. The device driver shares the additional tasks to support these options with the 82599.

7.2.4.1 Assumptions and Restrictions

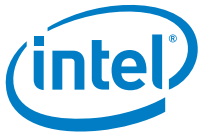
The following assumptions apply to the TCP / UDP segmentation implementation in the 82599:

- To limit the internal cache dimensions, software is required to spread the header onto a maximum four descriptors, while still allowed to mix header and data in the last header buffer. This limitation stands for up to Layer 4 header included, and for IPv4 or IPv6 independently.
- The maximum size of a single TSO can be as large as defined by the *PAYLEN* field in the Tx data descriptor (such as up to 256 KB).
- The *RS* bit operation is not changed. Interrupts are set after data in the buffers pointed to by individual descriptors is transferred (DMA'ed) to hardware.
- SNAP packets are supported for segmentation with the following restriction. The location of the 802.3 length field in 802.3+SNAP packets is at MACLEN minus eight bytes (MACLEN is indicated in the context descriptor).
- IP tunneled packets are not supported for offloading under TSO operation.
- Software must enable the Ethernet CRC offload in the HLREG0.TXCRCEN register since CRC must be inserted by hardware after the checksum has been calculated.
- Software must initialize the appropriate checksum fields in the packet’s header.

7.2.4.2 Transmission Process

The transmission process involves the following:

- The protocol stack receives from an application a block of data that is to be transmitted.
- The protocol stack calculates the number of packets required to transmit this block based on the MTU size of the media and required packet headers.
- The stack interfaces with the device driver and passes the block down with the appropriate header information: Ethernet, IP, optional IPSec and TCP / UDP headers.
- The stack interfaces with the device driver and commands the driver to send the individual packet. The device driver sets up the interface to the hardware (via descriptors) for the TCP / UDP segmentation.
- The hardware transfers (DMA's) the packet data and performs the Ethernet packet segmentation and transmission based on offset and payload length parameters in the TCP/IP or UDP/IP context descriptor including:
  - Packet encapsulation
  - Header generation and field updates including IPv4/IPv6 and TCP/UDP checksum generation.
- The driver returns ownership of the block of data to the NOS when the hardware has completed the DMA transfer of the entire data block.



7.2.4.2.1 TCP and UDP Segmentation Data Fetch Control

To perform TCP / UDP segmentation in the 82599, the DMA must be able to fit at least one packet of the segmented payload into available space in the on-chip packet buffer. The DMA does various comparisons between the remaining payload and the packet buffer available space, fetching additional payload and sending additional packets as space permits.

The 82599 enables interleaving between different TSO requests at an Ethernet packet level. In other words, the 82599 might fetch part of a TSO from a queue, equivalent to one or more Ethernet packets, then transition to another queue and fetch the equivalent of one or more packets (TSO or not), then move to another queue (or the first queue), etc. The 82599 decides on the order of data fetched based on its QoS requirements (such as bandwidth allocation and priority).

In order to enable interleaving between descriptor queues at the Ethernet frame resolution inside TSO requests, the frame header pointed by the so called header descriptors are re-read from system memory for every TSO segment (once per packet), storing in an internal cache only the header's descriptors instead of the header's content.

7.2.4.2.2 TCP and UDP Segmentation Write-back Modes

TCP / UDP segmentation mode uses the buffers that contain the header of the packet multiple times (once for each transmitted segment). Software should guarantee that the header buffers are available throughout the entire TSO transmission. Therefore, software should not re-use any descriptors of the TSO header during the TSO transmission.

7.2.4.3 TCP and UDP Segmentation Performance

Performance improvements for a hardware implementation of TCP / UDP segmentation offload include:

- The stack does not need to partition the block to fit the MTU size, saving CPU cycles.
- The stack only computes one Ethernet, IP, and TCP / UDP header per segment, saving CPU cycles.
- The stack interfaces with the device driver only once per block transfer, instead of once per frame.
- Larger PCI bursts are used, which improves bus efficiency (such as lowering transaction overhead).
- Interrupts are easily reduced to one per TCP / UDP message instead of one per packet.
- Fewer I/O accesses are required to command the hardware.

7.2.4.4 Packet Format

Typical TCP/IP transmit window size is 8760 bytes (about six full size frames). Today the average size on corporate Intranets is 12-14 KB, and normally the maximum window size allowed is 64 KB (unless Windows Scaling — RFC 1323 is specified). A TCP / UDP message can be as large as 256 KB and is generally fragmented across multiple pages in host memory. The 82599 partitions the data packet into standard Ethernet frames prior to transmission. The 82599 supports calculating the Ethernet, IP, TCP, and even UDP headers, include checksum, on a frame-by-frame basis.

Table 7-41 TCP/IP and UDP/IP Packet Format Sent by Host

Pseudo Header			Data
Ethernet	IPv4/IPv6	TCP/UDP	DATA (full TCP message)

Table 7-42 Packets Format Sent by Device

Pseudo Header (updated)	Data (first MSS)	FCS	...	Pseudo Header (updated)	Data (Next MSS)	FCS	...

Frame formats supported by the 82599 include:

- Ethernet 802.3
- IEEE 802.1Q VLAN (Ethernet 802.3ac)
- Ethernet Type 2
- Ethernet SNAP
- IPv4 headers with options
- IPv4 headers without options with one AH/ESP IPsec header
- IPv6 headers with extensions
- TCP with options
- UDP with options

VLAN tag insertion is handled by hardware.

Note: UDP (unlike TCP) is not a reliable protocol and fragmentation is not supported at the UDP level. UDP messages that are larger than the MTU size of the given network medium are normally fragmented at the IP layer. This is different from TCP, where large TCP messages can be fragmented at either the IP or TCP layers depending on the software implementation.

The 82599 has the ability to segment UDP traffic (in addition to TCP traffic); however, because UDP packets are generally fragmented at the IP layer, the 82599's segmentation capability might not be used in practice for UDP.