Table 7-3 Rx Queuing Schemes Supported with Virtualization

DCB	RSS	DCB / RSS Queues	Special Filters[1]
No	No	16 pools x 1 queue 32 pools x 1 queue 64 pools x 1 queue - Rx queue 0 of each pool	Supported
No	Yes[2]	32 pools x 4 RSS 64 pools x 2 RSS	Supported
Yes	No	16 pools x 8 TCs 32 pools x 4 TCs	Supported
Yes	Yes	Not supported	

1. Special filters include: L2 filters, FCoE redirection, SYN filter and L3/L4 5-tuble filters. When possible, it is recommended to assign Rx queues not used by DCB/RSS queues.
2. RSS might not be useful for IOV mode since the 82599 supports a single RSS table for the entire device.

Table 7-4 Queue Indexing Illustration in Virtualization Mode

Queue Index bits	6	5	4	3	2	1	0
VT(64) + RSS	VF Index						RSS
VT(32) + RSS	VF Index					RSS	
VT(16) + RSS	Not Supported						
VT(32) + DCB(4)	VF Index					TC	
VT(16) + DCB(8)	VF Index				TC		

Selecting a scheme is done in the following manner:

- Non-IOV mode
  - — Selected via the *Multiple Receive Queues Enable* field in the MRQC register.
- IOV mode
  - — Determine the number of pools: the number must support the value configured by the operating system in the PCIe NumVFs field (see Section 9.4.4.5). Therefore, the number of pools is min of {16, 32, 64} that is still >= NumVFs.
  - — Determine DCB mode via the *RT Enable* CSR field.
  - — Note that RSS is not supported in IOV mode since there is only a single RSS hash function in the hardware.

A received packet is assigned to an absolute queue index according to the ordering shown in Figure 7-7). It is software responsibility to define a queue that belongs to the matched pool.

- DCB and RSS filters — The supported modes are listed in Table 7-3 and detailed as follows. The associated queue indexes are listed in Table 7-4.
  - No DCB, No RSS — A single queue is allocated per pool with either 32 or 64 pools enabled. In 64 pools setting, queues '2xN'...'2xN+1' are allocated to pool 'N'; In 32 pools setting, queues '4xN'...'4xN+3' are allocated to pool 'N'.
  - RSS only — All 128 queues are allocated to pools. Several configurations are supported: 32 pools with 4 RSS queues each, and 64 pools with 2 queues each. Note that it is possible to use a subset of the RSS queues in each pool. The LSBits of the queue indexes are defined by the RSS index, and the pool index is used as the MS bits.
  - DCB only — All 128 queues are allocated to pools. Several configurations are supported: 16 pools with 8 TCs each, or 32 pools with 4 TCs each. The LSBits of the queue indexes are defined by the TC index, and the pool index is used as the MS bits.

When operating in conjunction with RSS, the number of RSS queues can vary per pool as defined by the PSRTYPE[n].RQPL. Each pool can be configured to a different number of RSS queues (0/1/2/4 queues) up to the maximum possible queues in the selected mode of operation. The output of the RSS redirection table is masked accordingly to generate an RSS index of the right width. When configured to less than the maximum number of queues, the respective MS bits of the RSS index are set to zero.

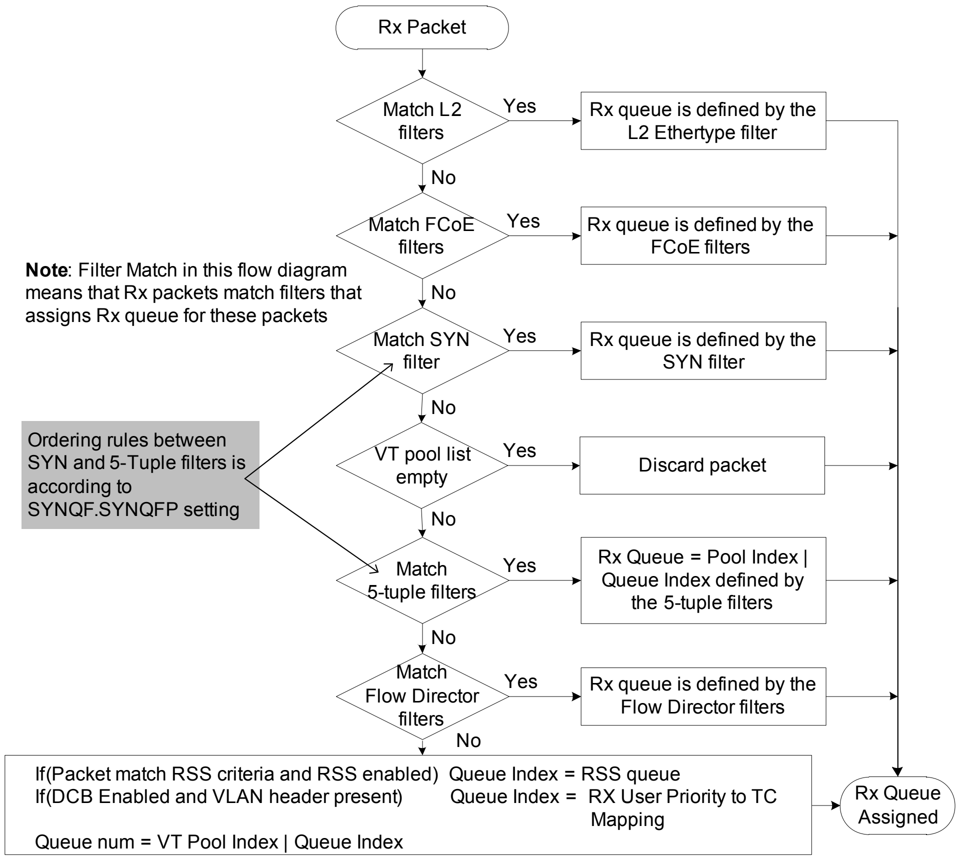
Rx Packet
Match L2 filters
Yes
Rx queue is defined by the L2 Ethertype filter
No
Match FCoE filters
Yes
Rx queue is defined by the FCoE filters
Note: Filter Match in this flow diagram means that Rx packets match filters that assigns Rx queue for these packets
No
Match SYN filter
Yes
Rx queue is defined by the SYN filter
No
Ordering rules between SYN and 5-Tuple filters is according to SYNQF.SYNQFP setting
VT pool list empty
Yes
Discard packet
No
Match 5-tuple filters
Yes
Rx Queue = Pool Index | Queue Index defined by the 5-tuple filters
No
Match Flow Director filters
Yes
Rx queue is defined by the Flow Director filters
No
If(Packet match RSS criteria and RSS enabled) Queue Index = RSS queue
If(DCB Enabled and VLAN header present) Queue Index = RX User Priority to TC Mapping
Queue num = VT Pool Index | Queue Index
Rx Queue Assigned

Figure 7-7 Rx Queuing Flow (Virtualization Case)

7.1.2.3 L2 Ethertype Filters

These filters identify packets by their L2 Ethertype, 802.1Q user priority and optionally assign them to a receive queue. The following possible usages have been identified at this time:

- DCB LLDP packets — Identifies DCB control packets
- IEEE 802.1X packets — Extensible Authentication Protocol (EAPOL) over LAN
- Time sync packets (such as IEEE 1588) — Identifies Sync or Delay_Req packets
- FCoE packets (possibly two UP values)
- The L2 type filters should not be set to IP packet type as this might cause unexpected results

The 82599 incorporates eight Ethertype filters defined by a set of two registers per filter: ETQF[n] and ETQS[n].

The L2 packet type is defined by comparing the *Ether-Type* field in the Rx packet with the ETQF[n].EType (regardless of the pool and UP matching). The *Packet Type* field in the Rx descriptor captures the filter number that matched with the L2 Ethertype. See Section 7.1.6.2 for a description of the *Packet Type* field.

The following flow is used by the Ethertype filters:

1. If the *Filter Enable* bit is cleared, the filter is disabled and the following steps are ignored.
2. Receive packet matches any ETQF filters if the *EtherType* field in the packet matches the *EType* field of the filter. Note that the following steps are ignored if the packet does not match the ETQF filters.
3. Packets that match any ETQF filters is a candidate for the host. If the packet also matches the manageability filters, it is directed to the host as well regardless of the MANC2H register setting.
4. If the *FCoE* field is set, the packet is identified as an FCoE packet.
5. If the *1588 Time Stamp* field is set, the packet is identified as an IEEE 1588 packet.
6. If the *Queue Enable* bit is cleared, the filter completed its action on the packet. Else, the filter is also used for queuing purposes as described in the sections that follow.
7. If the *Pool Enable* field is set, the *Pool* field of the filter determines the target pool for the packet. The packet can still be mirrored to other pools as described in Section 7.10.3. See the sections that follow for more details on the use of the *Pool* field.
8. The *RX Queue* field determines the destination queue for the packet. In case of a mirrored packet, only the copy of the packet that is targeted to the pool defined by the *Pool* field in the ETQF register is routed according to the *Rx Queue* field.

Setting the ETQF[n] registers is described as follows:

- The *Filter Enable* bit enables identification of Rx packets by Ethertype according to this filter. If this bit is cleared, the filter is ignored.
- The *EType* field contains the 16-bit Ethertype compared against all L2 type fields in the Rx packet.
- The FCoE bit indicates that the Ethertype defined in the *EType* field is an FCoE EType. Packets that match this filter are identified as FCoE packets.
- The *1588 Time Stamp* bit indicates that the Ethertype defined in the *EType* field is identified as IEEE 1588 EType. Packets that match this filter are time stamped according to the IEEE 1588 specification.
- The *Pool* field defines the target pool for a packet that matches the filter.
  — It applies only in virtualization modes. The pool index is meaningful only if the *Pool Enable* bit is set.
  — If the *Pool Enable* bit is set then the *Queue Enable* bit in the ETQS register must be set as well. In this case, the *Rx Queue* field in the ETQS must be part of the pool number defined in the ETQF.

Setting the ETQS[n] registers is described as follows:

- The *Queue Enable* bit enables routing of the Rx packet that match the filter to Rx queue as defined by the *Rx Queue* field.

- The *Rx Queue* field contains the destination queue (one of 128 queues) for the packet.
- The *Low Latency Interrupt* bit enables LL interrupt assertion by the Rx packet that matches this filter.

Special considerations for virtualization modes:

- Packets that match an Ethertype filter are diverted from their original pool (as defined by the VLAN and Ethernet MAC address filters) to the pool defined in the *Pool* field in the ETQF registers.
- The same applies for multicast packets. A single copy is posted to the pool defined by the filter.
- Mirroring rules
  - — A packet sent to a pool by an ETQF filter is still a candidate for mirroring using the standard mirroring rules.
  - — The Ethertype filter does not take part in the decision on the destination of the mirrored packet.

7.1.2.4 FCoE Redirection Table

The FCoE redirection table is a mechanism to distribute received FCoE packets into several descriptor queues. Software might assign each queue to a different processor, sharing the load of packet processing among multiple processors. The FCoE redirection table assigns Rx queues to packets that are identified as FCoE in the ETQF[n] registers but not assigned to queues in the ETQS[n] registers.

Figure 7-8 illustrates the computing of the assigned Rx queue index by the FCoE redirection table.

- The Rx packet is parsed extracting the OX_ID or the RX_ID depending on the *Exchange Context* in the *F_CTL* field in the FC header. At zero the RX_ID is used; at one the OX_ID is used.
- The three LSBits of the OX_ID or RX_ID are used as an address to the redirection table (FCRETA[n] register index).
- The FCoE redirection table is enabled by the FCRECTL.ENA bit. If enabled, the content of the selected FCRETA[n] register is the assigned Rx queue index.

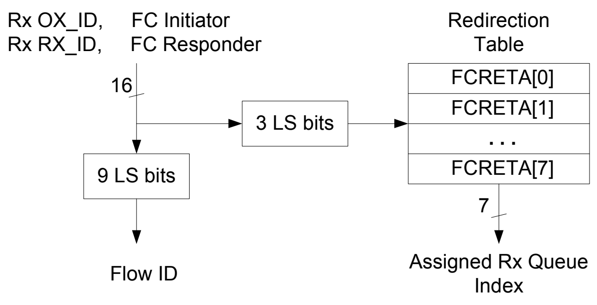

Figure 7-8 FCoE Redirection Table

7.1.2.5 L3/L4 5-tuple Filters

These filters identify specific L3/L4 flows or sets of L3/L4 flows and routes them to dedicated queues. Each filter consists of a 5-tuple (protocol, source and destination IP addresses, source and destination TCP/UDP/SCTP port) and routes packets into one of the Rx queues.

The 82599 incorporates 128 such filters, used also to initiate Low Latency Interrupts (LLI). The specific filtering rules are:

- Filtering rules for IPv6 packets:
  - If a filter defines at least one of the IP source and destination addresses, then an IPv6 packet always misses such a filter.
  - If a filter masks both the IP source and destination addresses, then an IPv6 packet is compared against the remaining fields of the filter.
- Packets with tunneling (any combination of IPv4 and IPv6) miss the 5-tuple filters.
- Fragmented packets miss the 5-tuple filters.

In a virtualized environment, any 5-tuple filters is associated with a unique pool:

- The packet must first match the L2 filters described in Section 7.10.3.3 and Section 7.10.3.4. The outcome of the L2 filters is a set of pool values associated with the packet. The *Pool* field of the 5-tuple filter is then compared against the set of pools to which the packet is steered. A filter match is considered only to the indicated pool in the filter.

If a packet matches more than one 5-tuple filter, then:

- For queuing decision — The priority field identifies the winning filter and therefore the destination queue.
- For queuing decision — If the packet matches multiple filters with the same priority, the filters with the lower index takes affect.
- For Low Latency Interrupt (LLI) — An LLI is issued if one or more of the matching filters are set for LLI.

The 5-tuple filters are configured via the FTQF, SDPQF, L34TIMIR, DAQF, and SAQF registers, as follows (described by filter):

- Protocol — Identifies the IP protocol, part of the 5-tuple. Enabled by a bit in the mask field. Supported protocol fields are TCP, UDP, SCTP or other (neither TCP nor UDP nor SCTP).
- Source address — Identifies the IP source address, part of the 5-tuple. Enabled by a bit in the mask field. Only IPv4 addresses are supported.
- Destination address — Identifies the IP destination address, part of the 5-tuple. Enabled by a bit in the mask field. Only IPv4 addresses are supported.
- Source port — Identifies the TCP/UDP/SCTP source port, part of the 5-tuple. Enabled by a bit in the mask field.
- Destination port — Identifies the TCP/UDP/SCTP destination port, part of the 5-tuple queue filters. Enabled by a bit in the mask field.
- Queue Enable — Enables the packets routing to queues based on the Rx Queue index of the filter.

- Rx Queue — Determines the Rx queue for packets that match this filter.
- Pool — Applies only in the virtualized case (while *Pool Mask* bit = 0b). This field must match one of the pools enabled for this packet in the L2 filters.
  - In non-virtualized case the *Pool Mask* bit must be set to 1b.
  - In the virtualized case, the pool must be defined (*Pool Mask* = 0b and *Pool* = valid index). The *Rx Queue* field defines the absolute queue index. In case of mirroring or replication, only the copy of the packet destined to the matched pool in the filter is routed according to the *Rx Queue* field.
- Mask — A 5-bit field that masks each of the fields in the 5-tuple (L4 protocol, IP addresses, TCP/UDP ports). The filter is a logical AND of the non-masked 5-tuple fields. If all 5-tuple fields are masked, the filter is not used for queue routing.
- Priority — A 3-bit field that defines one of seven priority levels (001b-111b), with 111b as the highest priority. Software must insure that a packet never matches two or more filters with the same priority value.

Note: There are 128 different 5-tuple filter configuration registers sets, with indexes [0] to [127]. The mapping to a specific Rx queue is done by the *Rx Queue* field in the L34TIMIR register, and not by the index of the register set.

7.1.2.6 SYN Packet Filters

The 82599 might route TCP packets whose SYN flag is set into a separate queue. SYN packets are used in SYN attacks to load the system with numerous requests for new connections. By filtering such packets to a separate queue, security software can monitor and act on SYN attacks.

The following rules apply:

- A single SYN filter is provided.

The SYN filter is configured via the SYNQF register as follows:

- The *Queue Enable* bit enables SYN filtering capability.
- The *Rx Queue* field contains the destination queue for the packet (one of 128 queues). In case of mirroring (in virtualization mode), only the original copy of the packet is routed according to this filter.

7.1.2.7 Flow Director Filters

The flow director filters identify specific flows or sets of flows and routes them to specific queues. The flow director filters are programmed by FDIRCTRL and all other FDIR registers. The 82599 shares the Rx packet buffer for the storage of these filters. Basic rules for the flow director filters are:

- IP packets are candidates for the flow director filters (meaning non-IP packets miss all filters)
- Packets with tunneling (any combination of IPv4 and IPv6) miss all filters
- Fragmented packets miss all filters

- In VT mode, the *Pool* field in FDIRCMD must be valid. If the packet is replicated, only the copy that goes to the pool that matches the *Pool* field is impacted by the filter.

The flow director filters cover the following fields:

- VLAN header
- Source IP and destination IP addresses
- Source port and destination port numbers (for UDP and TCP packets)
- IPv4 / IPv6 and UDP / TCP or SCTP protocol match
- Flexible 2-byte tuple anywhere in the first 64 bytes of the packet
- Target pool number (relevant only for VT mode)

Note: IPv6 extended headers are parsed by the 82599, enabling TCP layer header recognition. Still the IPv6 extended header fields are not taken into account for the queue classification by Flow Director filter. This rule do not apply for security headers and fragmentation header. Packets with fragmentation header miss this filter. Packets with security extended headers are parsed only up to these headers and therefore can match only filters that do not require fields from the L4 protocol.

The 82599 support two types of filtering modes (static setting by the FDIRCTRL.Perfect-Match bit):

- Perfect match filters — The hardware checks a match between the masked fields of the received packets and the programmed filters. Masked fields should be programmed as zeros in the filter context. The 82599 support up to 8 K - 2 perfect match filters.
- Signature filters — The hardware checks a match between a hash-based signature of the masked fields of the received packet. The 82599 supports up to 32 K - 2 signature filters.
- Notation — The *Perfect Match* fields and *Signature* field are denoted as *Flow ID* fields.

The 82599 supports masking / range for the previously described fields. These masks are defined globally for all filters in the FDIR…M register.

- The following fields can be masked per bit enabling power of two ranges up to complete enable / disable of the fields: IPv4 addresses and L4 port numbers.
- The following fields can be masked per byte enabling lower granularity ranges up to complete enable / disable of the fields: IPv6 addresses. Note that in perfect match filters the destination IPv6 address can only be compared as a whole (with no range support) to the IP6AT.
- The following fields can be either enabled or disabled completely for the match functionality: VLAN ID tag; VLAN Priority + CFI bit; Flexible 2-byte tuple and target pool. Target pool can be enabled by software only when VT is enabled as well.

Flow director filters have the following functionality in virtualization mode:

- Flow director filters are programmed by the registers in the PF described in Section 7.1.2.7.11 and Section 7.1.2.7.12.

7.1.2.7.1 Flow Director Filters Actions

Flow director filters might have one of the following actions programmed per filter in the FDIRCTRL register:

- Drop packet or pass to host as defined by the *Drop* bit.
  - Matched packets to a flow director filter is directed to the assigned Rx queue only if the packet does not match the L2 filters for queue assignment nor the SYN filter for queue assignment nor the 5-tuple filters for queue assignment.
  - Packets that match pass filters are directed to the Rx queue defined in the filter context as programmed by the FDIRCMD.Rx-Queue. In a non-VT setting, the *Rx Queue* field defines the absolute queue number. In VT setting, the *Rx Queue* field defines the relative queue number within the pool.
  - Packets that match drop filters are directed to the Rx queue defined per all filters in the FDIRCTRL.DROP-Queue. The 82599 drops these packets if software does not enable the specific Rx queue.
- Trigger low latency interrupt is enabled by the *INT* bit.
  - Matched packets to a flow director filter can generate LLI if the packet does not match the L2 filters for queue assignment nor the SYN filter for queue assignment nor the 5-tuple filters for queue assignment.

7.1.2.7.2 Flow Director Filters Status Reporting

Shared status indications for all packets:

- The 82599 increments the FDIRMATCH counter for packets that match a flow director filter. It also increments the FDIRMISS counter for packets that do not match any flow director filter.
- The *Flow Director Filter Match* (*FLM*) bit in the *Extended Status* field of the Rx descriptor is set for packets that match a flow director filter.
- The flow ID parameters are reported in the *Flow Director Filter ID* field in the Rx descriptor if enabled by the FDIRCTRL.Report-Status. When the *Report-Status* bit is set, the RXCSUM.PCSD bit should be set as well. This field is indicated for all packets that match or do not match the flow director filters. Note that it is required to set the FDIRCTRL.Report-Status bit to enable the FLM status indication as well as any Flow Director error indications in the receive descriptor.
  - For packets that do not match a flow director filter, if the FDIRCTRL.REPORT_STATUS_ALWAYS is set, the Flow Director Filter ID field can be used by software for future programming of a matched filter. Otherwise, the RSS hash value is reported.
  - For packets that match a flow director filter, the *Flow Director Filter ID* field can be used by software to identify the flow of the Rx packet.

Too long linked list exception (linked list and too long terms are illustrated in Figure 7-9):

- The maximum recommended linked list length is programmed in the FDIRCTRL.Max-Length field
- The length exception is reported in the *FDIRErr* field in the Rx descriptor

- Packets that do not match any flow director filter, reports this exception if the length of the existing linked list is already at the maximum recommended length. Software can use it to avoid further programming of additional filters to this linked list before other filters are removed.
- Packets that match a pass filter report this exception if the distance of the matched filter from the beginning of the linked list is higher than the above recommended length.
- Packets that match a drop filter are posted to the Rx queue programmed in the filter context instead of the global FDIRCTRL.Rx-Queue. The drop exception is reported in addition to the length exception (in the same field in the Rx descriptor).

Collision exception:

- Packets that matches a collided filter report this exception in the *FDIRErr* field in the Rx descriptor.
- Collision events for signature-based filters should be rare. Still it might happen because multiple flows can have the same hash and signature values. Software might leave the setting as is while the collided flows are handled according to the actions of the first programmed flow. On the other hand, software might choose to resolve the collision by programming the collided flows in the 5-tuples filters. Only one flow (out of the collided ones) might remain in the flow director filters. In order to clear the collision indication in the programmed filter, software should remove the filter and then re-program it once again.
- Collision events for a perfect match filter should never happen. A collision error might indicate a programming fault that software might decide to fix.

7.1.2.7.3 Flow Director Filters Block Diagram

The following figure shows a block diagram of the flow director filters. Received flows are identified to buckets by a hash function on the relevant tuples as defined by the FDIR...M registers. Each bucket is organized in a linked list indicated by the hash lookup table. Buckets can have a variable length while the last filter in each bucket is indicated as a last. There is no upper limit for a linked list length during programming; however, a received packet that matches a filter that exceeds the FDIRCTRL.Max-Length are reported to software (see Section 7.1.2.7.5).

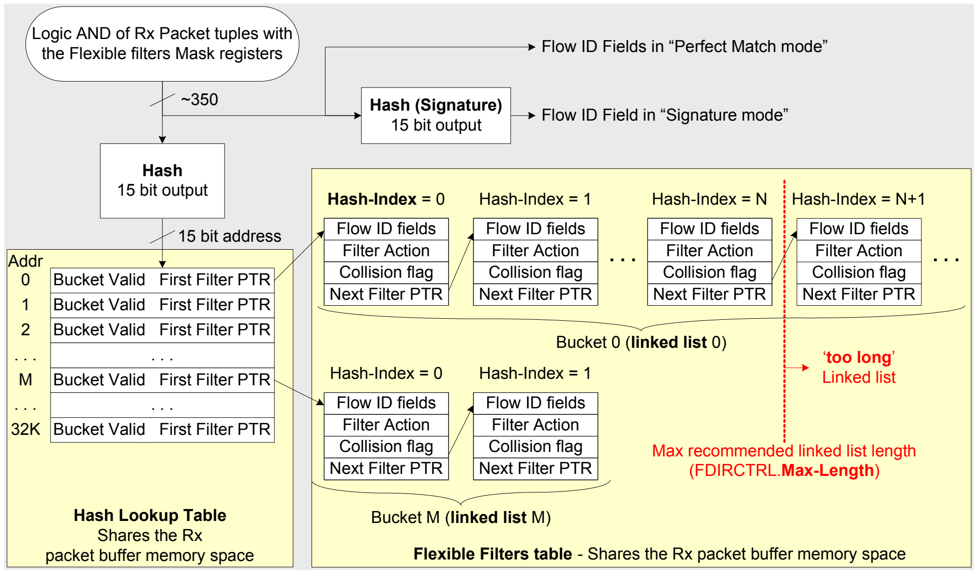
Logic AND of Rx Packet tuples with the Flexible filters Mask registers
~350
Flow ID Fields in "Perfect Match mode"
Hash (Signature) 15 bit output
Flow ID Field in "Signature mode"
Hash 15 bit output
15 bit address
Addr
0 Bucket Valid First Filter PTR
1 Bucket Valid First Filter PTR
2 Bucket Valid First Filter PTR
M Bucket Valid First Filter PTR
32K Bucket Valid First Filter PTR
Hash Lookup Table Shares the Rx packet buffer memory space
Hash-Index = 0
Hash-Index = 1
Hash-Index = N
Hash-Index = N+1
Flow ID fields
Filter Action
Collision flag
Next Filter PTR
Bucket 0 (linked list 0)
'too long' Linked list
Bucket M (linked list M)
Max recommended linked list length (FDIRCTRL.Max-Length)
Flexible Filters table - Shares the Rx packet buffer memory space

Figure 7-9 Flow Director Filters Block Diagram

7.1.2.7.4 Rx Packet Buffer Allocation

Flow director filters can consume zero space (when disabled) up to ~256 KB of memory. As shown in Figure 7-9, flow director filters share the same memory with the Rx packet buffer. Setting the *PBALLOC* field in the FDIRCTRL register, the software might enable and allocate memory for the flow director filters. The memory allocated to reception is the remaining part of the Rx packet buffer.

Table 7-5 Rx Packet Buffer Allocation

PBALLOC (2)	Effective Rx Packet Buffer Size (see following note)	Flow Director Filters Memory	Supported Flow Director Filters			
			Signature		Perfect Match	
			Filters	Bucket Hash	Filters	Bucket Hash
00 Flow Director is disabled	512 KB	0	0	n/a	0	n/a
01	448 KB	64 KB	8 K - 2	13 bits	2 K - 2	11 bits
10	384 KB	128 KB	16 K - 2	14 bits	4 K - 2	12 bits
11	256 KB	256 KB	32 K - 2	15 bits	8 K - 2	13 bits

Note: It is the user responsibility to ensure that sufficient buffer space is left for reception. The required buffer space for reception is a function of the number of traffic classes, flow control threshold values and remaining buffer space in between the thresholds. If flow director is enabled (such as PBALLOC > 0), software should set the RXPBSIZE[n] registers according to the total remaining part of the Rx packet buffer for reception.

For example, if PBALLOC equals one and there is only one buffer in the system, software should set RXPBSIZE[0] to 0x70000 (448 K) and RXPBSIZE[1...7] to zero. Another example is if PBALLOC equals two and DCB is enabled with four traffic classes then software might set RXPBSIZE[0...3] to 0x18000 (96 K) and RXPBSIZE[4...7] to zero.

Refer to Section 3.7.7.3.2 through Section 3.7.7.3.5 for recommended setting of the Rx packet buffer sizes and flow control thresholds.

7.1.2.7.5 Flow Director Filtering Reception Flow

- Rx packet is digested by the filter unit which parse the packet extracting the relevant tuples for the filtering functionality.
- The 82599 calculates a 15-bit hash value out of the masked tuples (logic mask of the tuples and the relevant mask registers) using the hash function described in Section 7.1.2.7.15.
- The address in the hash lookup table points to the selected linked list of the flow director filters.
- The 82599 checks the *Bucket Valid* flag. If it is inactive, then the packet does not match any filter. Otherwise, *Bucket Valid* flag is active, proceed for the next steps.
- The 82599 checks the linked list until it reaches the last filter in the linked list or until a matched filter is found.
- Case 1: matched filter is found:
  - Increment the FDIRMATCH statistic counter.
  - Process the filter's actions (queue assignment and LLI) according to queue assignment priority. Meaning, the actions defined in this filter takes place only if the packet did not match any L2 filter or SYN filter or 5-tuple filter that assigns an Rx queue to the packet.
  - Rx queue assignment according to the filter context takes place if *Queue-EN* is set. In VT mode, the Rx queue in the filter context defines a relative queue within the pool.
  - LLI is generated if the *INT* bit is set in the filter context.
  - Post the packet to host including the flow director filter match indications as described in Section 7.1.2.7.2.
- Case 2: matched filter is not found:
  - Increment the FDIRMISS statistic counter.
  - Post the packet to host including the flow director filter miss indications as described in Section 7.1.2.7.2.

7.1.2.7.6 Add Filter Flow

The software programs the filters parameters in the registers described in Section 7.1.2.7.12 and Section 7.1.2.7.13 while keeping the FDIRCMD.Filter-Update bit inactive. As a result, the 82599 checks the bucket valid indication in the hash lookup table (that matches the FDIRHASH.Hash) for the presence of an existing linked list.

Following are the two programming flows that handle a presence of an existing linked list or creating a new linked list.

- Case 1: Add a filter to existing linked list:

  The 82599 checks the linked list until it reaches the last filter in the list or until a matched filter is found. Handle the filter programming in one of the following cases:

  — Matched filter is found (equal flow ID) with the same action parameters — The programming is discarded silently. This is a successful case since the programmed flow is treated as requested.

  — Matched filter is found (equal flow ID) with different action parameters — The 82599 keeps the old setting of the filter while setting the *Collision* flag in the filter context and increments the COLL counter in the FDIRFREE register (see Section 7.1.2.7.2 for software handling of collision during packet reception).

  — Matched filter is found (equal flow ID) with different action parameters and the *Collision* flag is already set — The programming is discarded silently. Software gets the same indications as the previous case.

  — Matched filter is not found (no collision) — The 82599 checks for a free space in the flow director filters table.

  — No space case — Discard programming; increment the FADD counter in the FDIRFSTAT register and assert the flow director interrupt. Following this interrupt software should read the FDIRFSTAT register and FDIRFREE.FREE field, for checking the interrupt cause.

  — Free space is found — Good programming case: Add the new filter at the end of the linked list while indicating it as the last one. Program the *Next Filter PTR* field and then clear the *Last* flag in the filter that was previously the last one.

- Case 2 — Create a new linked list:

  The 82599 looks for an empty space in the flow director filters table:

  — Handle no empty space the same as in Case 1.

  — Good programming case: Add the new filter while indicating it as the last one in the linked list. Then, program the hash lookup table entry by setting the *Valid* flag and the *First Filter PTR* pointing to the new programmed filter.

Additional successful add flow indications:

- Increment the ADD statistic counter in the FDIRUSTAT register.
- Reduce the FREE counter in the FDIRFREE register and then indicate the number of free filters. If the FREE counter crosses the full-thresh value in the FDIRCTRL register, then assert the flow director filter interrupt. Following this interrupt software should read the FDIRFSTAT register and FDIRFREE.FREE field, for checking the interrupt cause.
- Compare the length of the new linked list with MAXLEN in the FDIRLEN register. If the new linked list is longer than MAXLEN, update the FDIRLEN by the new flow.

Note: The 82599 also reports the number of collided filters in FDIRFREE.COLL. Software might monitor this field periodically as an indication for the filters efficiency.

7.1.2.7.7 Update Filter Flow

In some applications, it is useful to update the filter parameters, such as the destination Rx queue. Programing filter parameters is described in Section 7.1.2.7.6.

Setting the *Filter-Update* bit in the FDIRCMD register has the following action:

- Case 1: Matched filter does not exist in the filter table — Setting the *Filter-Update* bit has no impact and the command is treated as add filter.
- Case 2: Matched filter already exists in the filter table — Setting the *Filter-Update* bit enables filter parameter's update while keeping the collision indication as is.

The Update Filter Flow process requires internal memory space used to store temporary data until the update concludes. Therefore, Update Filter Flow can be used only if the maximum number of allocated flow director filters (as defined by FDIRCTRL.PBALLOC) is not fully used.

For example, if FDIRCTRL.PBALLOC=01b, memory is allocated for 2K-1 perfect filters. In this case, the Update Filter Flow can be used only if not more than 2K-2 filters were programmed.

7.1.2.7.8 Remove Filter Flow

Software programs the filter Hash and Signature / Software-Index in the FDIRHASH register. It then should set the FDIRCMD.CMD field to *Remove Flow*. Software might use a single 64-bit access to the two registers for atomic operation. As a result, the 82599 follows these steps:

- Check if such a filter exists in the flow director filters table.
- If there is no flow, then increment the FREMOVE counter in the FDIRFSTAT register and skip the next steps.
- If the requested filter is the only filter in the linked list, then invalidate its entry in the hash lookup table by clearing the *Valid* bit.
- Else, if the requested filter is the last filter in the linked list, then invalidate the entry by setting the *Last* flag in the previous filter in the linked list.
- Else, invalidate its entry by programming the Next Filter PTR in the previous filter in the linked list, pointing it to the filter that was linked to the removed filter.

Additional indications for successful filter removal:

- Increment the remove statistic counter in the FDIRUSTAT register.
- Increment the FREE counter in the FDIRFREE register.

7.1.2.7.9 Remove all Flow Director Filters

In some cases there is a need to clear the entire flow director table. It might be useful in some applications that might cause the flow director table becoming too occupied. Then, software might clear the entire table enabling its re-programming with new active flows.

Following are steps required to clear the flow director table:

- Poll the FDIRCMD.CMD until it is zero indicating any previous pending commands to the flow director table is completed (at worst case the FDIRCMD.CMD should be found cleared on the second read cycle). Note that the software must not initiate any additional commands (add / remove / query) before this step starts and until this flow completes.
- Clear the FDIRFREE register (set the *FREE* field to 0x8000 and *COLL* field to zero).
- Set FDIRCMD.CLEARHT to 1b and then clear it back to 0b
- Clear the FDIRHASH register to zero
- Re-write FDIRCTRL by its previous value while clearing the *INIT-Done* flag.
- Poll the *INIT-Done* flag until it is set to one by hardware.
- Clear the following statistic registers: FDIRUSTAT; FDIRFSTAT; FDIRMATCH; FDIRMISS; FDIRLEN (note that some of these registers are read clear and some are read write).

7.1.2.7.10 Flow Director Filters Initializing Flow

Following a device reset, the flow director is enabled by programming the FDIRCTRL register, as follows:

- Set PBALLOC to non-zero value according to the required buffer allocation to reception and flow director filter (see Section 7.1.2.7.4). All other fields in the register should be valid as well (according to required setting) while the FDIRCTRL register is expected to be programmed by a single cycle. Any further programming of the FDIRCTRL register with non-zero value PBALLOC initializes the flow director table once again.
- Poll the *INIT-Done* flag until it is set to one by hardware (expected initialization flow should take about 55 μs at 10 Gb/s and 550 μs at 1 Gb/s (it is 5.5 ms at 100 Mb/s; however, this speed is not expected to be activated unless the 82599 is in a sleep state).

7.1.2.7.11 Query Filter Flow

Software might query specific filter settings and bucket length using the Query command.

- Program the filter Hash and Signature/Software-Index in the FDIRHASH register and set the *CMD* field in the FDIRCMD register to 11b (Query Command). A single 64-bit access can be used for this step.
- As a result, the 82599 provides the query result in the FDIRHASH, FDIRCMD and FDIRLEN registers (described in the sections as follows).
- Hardware indicates query completion by clearing the FDIRCMD.CMD field. The following table lists the query result.

Query Outcome	FDIRHASH -> Bucket Valid	FDIRCMD -> Filter Valid	FDIRLEN -> Bucket Length	FDIRCMD -> Filter ID Fields	FDIRCMD -> Filter Action
Empty Bucket	0	0	N/A	N/A	N/A
Valid Bucket, Matched Filter Not Found	1	0	Bucket linked list length	N/A	N/A
Found Signature Filter	1	1	Filter index within the linked list	0	Filter's parameters
Found Perfect Match Filter	1	1	Filter index within the linked list	Filter's parameters	Filter's parameters

7.1.2.7.12 Signature Filter Registers

The signature flow director filter is programmed by setting the FDIRHASH and FDIRCMD registers. These registers are located in consecutive 8-byte aligned addresses. Software should use a 64-bit register to set these two registers in a single atomic operation. Table 7-6 lists the recommended setting.

Table 7-6 Signature Match Filter Parameters

Filter Bucket Parameters — FDIRHASH	
Hash	Hash function used to define a bucket of filters. This parameter is part of the flow director filter ID that can be reported in the Rx descriptor. The size of this field can be 15 bits, 14 bits or 13 bits as explained in Section 7.1.2.7.4. Non-used upper bits (MS bits) should be set to zero.
Valid	Should be set to 1b.
Flow ID — FDIRHASH	
Signature	16-bit hash function used as the flow matching field. This parameter is also part of the flow director filter ID that can be reported in the Rx descriptor.
FDIRCMD — Programming Command and Filter action — Set Section 8.2.3.21.22 for all fields descriptions.	

7.1.2.7.13 Perfect Match Filter Registers

Perfect match filters are programmed by the following registers: FDIRSIPv6[n]; FDIRVLAN; FDIRPORT; FDIRIPDA; FDIRIPSA; FDIRHASH; FDIRCMD. Setting the FDIRCMD register, generates the actual programming of the filter. Therefore, write access to this register must be the last cycle after all other registers contain a valid content. Table 7-7 lists the recommended setting.

Note: Software filter programming must be an atomic operation. In a multi-core environment, software must ensure that all registers are programmed in a sequence with no possible interference by other cores.

Table 7-7 Perfect Match Filter Parameters

Filter Bucket Parameters and Software Index — FDIRHASH	
Hash	Hash function used to define a bucket of filters. This parameter is part of the flow director filter ID that can be reported in the Rx descriptor. The size of this field can be 13 bits, 12 bits or 11 bits as explained in Section 7.1.2.7.4. Non-used upper bits (MS bits) should be set to zero.
Valid	Should be set to 1b.
Software-Index	15-bit index provided by software at filter programming used by software to identify the matched flow. This parameter is also part of the flow director filter ID that can be reported in the Rx descriptor. *Note:* The Software-Index is used as the filter identifier. Therefore, it must be within the range of supported filters while any filter must have a single unique Software-Index value.
FDIRCMD — Programming Command and Filter Action See Section 8.2.3.21.22 for All Fields Descriptions	
Flow ID — Perfect Match Flow ID Parameters are Listed in the Following Registers and Fields	
FDIRSIPv6[0…2].IP6SA	Three MS DWord of the source IPv6. Meaningful for IPv6 flows depending on the FDIRIP6M.SIPM setting.
FDIRVLAN.VLAN	VLAN fields are meaningful depending on the FDIRM.VLANID and FDIRM.VLANP setting.
FDIRVLAN.FLEX	Flexible 2-byte field at offset FDIRCTRL.Flex-Offset. Meaningful depending on FDIRM.FLEX setting.
FDIRPORT.Source	L4 source port. Meaningful for TCP and UDP packets depending on the FDIRTCPM.SportM and FDIRUDPM.SportM setting.
FDIRPORT.Destination	L4 destination port. Meaningful for TCP and UDP packets depending on the FDIRTCPM.SportM and FDIRUDPM.SportM setting.
FDIRIPDA.IP4DA	IPv4 destination address. Meaningful depending on the FDIRDIP4M.IP-EN setting.
FDIRIPSA.IP4SA	IPv4 source address or LS DWord of the source IPv6 address. Meaningful for IPv4 flows depending on the FDIRSIP4M.IP-EN setting and for IPv6 flows depending on the FDIRIP6M.SIPM setting.

7.1.2.7.14 Multiple CPU Cores Considerations

Perfect match filters programming and any query cycles requires access to multiple registers. In order to avoid races between multiple cores, software might need to use one of the following programming methods:

- Use a software-based semaphore between the multiple cores for gaining control over the relevant CSR registers for complete programming or query cycles.
- Manage all programming and queries of the flow director filters by a single core.

Programming signature filters requires only the FDIRHASH and FDIRCMD registers. These two registers are located in 8-byte aligned adjacent addresses. Software could use an 8-byte register for the programming of these registers in a single atomic operation, which avoids the need for any semaphore between multiple cores.

7.1.2.7.15 Flow Director Hash Function

The 82599 supports programmable 16-bit hash functions based on two 32-bit keys, one for the lookup table identifying a bucket of filters and another one for the signature (FDIRHKEY and FDIRSKEY). The hash function is described in the sections that follow. In some cases, a smaller hash value than 16 bits is required. In such cases, the LS bits of the hash value are used.

```
    For (i=0 to 350) { if (Ext_K[i]) then Hash[15 : 0] = Hash[15 : 0] XOR Ext_S[15+i
: i] }
```

While using the following notations:

```
    'XOR' - Bitwise XOR of two equal length strings

    If ( xxx ) - Equals ‘true’ if xxx = ‘1’ and equals ‘false’ if xxx = ‘0’

    S[335:0] - The input bit string of the flow director tuples: 42 bytes listed
    in Table 7-8 AND-logic with the filters masks.

    Ext_S[n] - S[14:0] | S[335:0] | S[335:321] // concatenated

    K[31:0] - The hash key as defined by the FDIRHKEY or FDIRSKEY registers.

    Tmp_K[11*32-1:0] - (Temp Key) equals K[31:0] | K[31:0] ... // concatenated Key 11
times

    Ext_K[350:0] - (Extended Key) equals Tmp_K[351:1]
```

The input bit stream for the hash calculation is listed in the Table 7-8 while byte 0 is the MSByte (first on the wire) of the VLAN, byte 2 is the MSByte of the source IP (IPv6 case) and so on.

Table 7-8 Input Bit Stream for Hash Calculation

Bytes	Field
Bytes 0…1	VLAN tag
Bytes 2…17	Source IP (16 bytes for IPv6; 12 bytes of zero's | source IP for IPv4)
Bytes 18…33	Destination IP (16 bytes for IPv6; 12 bytes of zero's | source IP for IPv4)
34…37	L4 source port number | L4 destination port number Meaningful for TCP and UDP packets and zero bytes for SCTP packets
38…39	Flexible bytes
40	00b | pool number (as defined by FDIRCMD.Pool)
41	00000b | IPv6/IPv4 type | L4 type (as defined by FDIRCMD.IPV6 and FDIRCMD.L4TYPE, respectively)

7.1.2.8 Receive-Side Scaling (RSS)

RSS is a mechanism to distribute received packets into several descriptor queues. Software then assigns each queue to a different processor, therefore sharing the load of packet processing among several processors.

As described in Section 7.1, the 82599 uses RSS as one ingredient in its packet assignment policy (the others are the various filters, DCB and virtualization). The RSS output is an RSS index. The 82599 global assignment uses these bits (or only some of the LSBs) as part of the queue number.

Figure 7-10 shows the process of computing an RSS output:

1. The receive packet is parsed into the header fields used by the hash operation (such as IP addresses, TCP port, etc.)
2. A hash calculation is performed. The 82599 supports a single hash function, as defined by MSFT RSS. The 82599 therefore does not indicate to the device driver which hash function is used. The 32-bit result is fed into the packet receive descriptor.
3. The seven LSBs of the hash result are used as an index into a 128-entry redirection table. Each entry provides a 4-bit RSS output index.

When RSS is enabled, the 82599 provides software with the following information as:

1. Required by Microsoft (MSFT) RSS
2. Provided for device driver assist:

- A Dword result of the MSFT RSS hash function, to be used by the stack for flow classification, is written into the receive packet descriptor (required by MSFT RSS).
- A 4-bit RSS *Type* field conveys the hash function used for the specific packet (required by MSFT RSS).

Enabling rules:

- RSS is enabled in the MRQC register.
- RSS enabling cannot be done dynamically while it must be preceded by a software reset.
- RSS status field in the descriptor write-back is enabled when the RXCSUM.PCSD bit is set (fragment checksum is disabled). RSS is therefore mutually exclusive with UDP fragmentation checksum offload.
- Support for RSS is not provided when legacy receive descriptor format is used.

Disabling rules:

- Disabling RSS on the fly is not allowed, and the 82599 must be reset after RSS is disabled.
- When RSS is disabled, packets are assigned an RSS output index = zero.

When multiple request queues are enabled in RSS mode, un-decodable packets are assigned an RSS output index = zero. The 32-bit tag (normally a result of the hash function) equals zero.

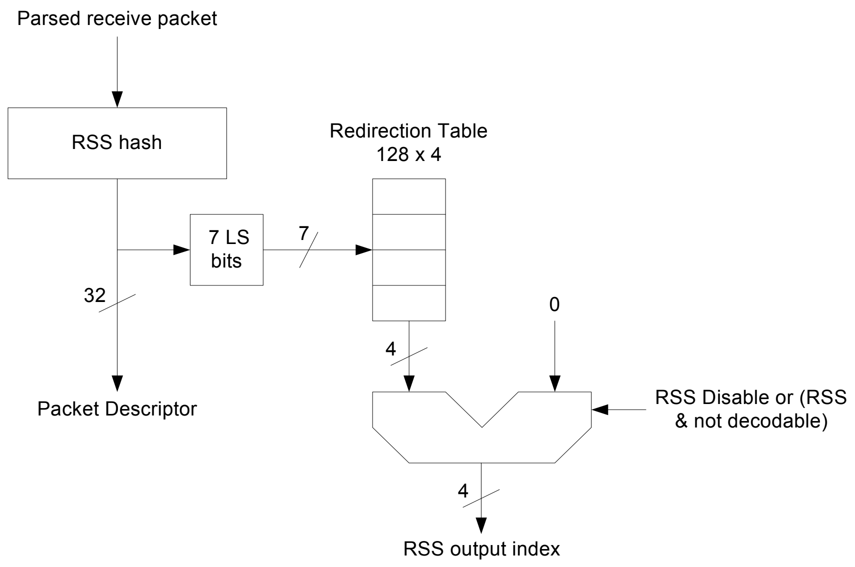

Figure 7-10 RSS Block Diagram

7.1.2.8.1 RSS Hash Function

This section provides a verification suite used to validate that the hash function is computed according to MSFT nomenclature.

The 82599's hash function follows the MSFT definition. A single hash function is defined with several variations for the following cases:

- TcpIPv4 — The 82599 parses the packet to identify an IPv4 packet containing a TCP segment per the following criteria. If the packet is not an IPv4 packet containing a TCP segment, RSS is not done for the packet.
- IPv4 — parses the packet to identify an IPv4 packet. If the packet is not an IPv4 packet, RSS is not done for the packet.
- TcpIPv6 — parses the packet to identify an IPv6 packet containing a TCP segment per the following criteria. If the packet is not an IPv6 packet containing a TCP segment, RSS is not done for the packet.
- IPv6 — The 82599 parses the packet to identify an IPv6 packet. If the packet is not an IPv6 packet, RSS is not done for the packet.

Note: Tunneled IP to IP packets are considered for the RSS functionality as IP packets. The RSS logic ignores the L4 header while using the outer (first) IP header for the RSS hash.

The following additional cases are not part of the MSFT RSS specification:

- UdpIPV4 — The 82599 parses the packet to identify a packet with UDP over IPv4.
- UdpIPV6 — The 82599 parses the packet to identify a packet with UDP over IPv6.

A packet is identified as containing a TCP segment if all of the following conditions are met:

- The transport layer protocol is TCP (not UDP, ICMP, IGMP, etc.).
- The TCP segment can be parsed (such as IPv4 options or IPv6 extensions can be parsed, packet not encrypted, etc.).
- The packet is not fragmented (even if the fragment contains a complete L4 header).

Note: IPv6 extended headers are parsed by the 82599, enabling TCP layer header recognition. Still the IPv6 extended header fields are not taken into account for the queue classification by RSS filter. This rule do not apply for security headers and fragmentation header. Packets with fragmentation header miss this filter. Packets with security extended headers are parsed only up to these headers and therefore can match only filters that do not require fields from the L4 protocol.

Bits[31:16] of the Multiple Receive Queues Command (MRQC) register enable each of the above hash function variations (several might be set at a given time). If several functions are enabled at the same time, priority is defined as follows (skip functions that are not enabled):

- IPv4 packet
  - Try using the TcpIPv4 function
  - Try using UdpIPv4 function
  - Try using the IPv4 function
- IPv6 packet
  - Try using the TcpIPv6 function.
  - Try using UdpIPv6 function.
  - Try using the IPv6 function

The following combinations are currently supported:

- Any combination of IPv4, TcpIPv4, and UdpIPv4.

And/Or:

- Any combination of either IPv6, TcpIPv6, and UdpIPv6.

When a packet cannot be parsed by the previous rules, it is assigned an RSS output index = zero. The 32-bit tag (normally a result of the hash function) equals zero.

The 32-bit result of the hash computation is written into the packet descriptor and also provides an index into the redirection table.

The following notation is used to describe the following hash functions:

- Ordering is little endian in both bytes and bits. For example, the IP address 161.142.100.80 translates into 0xa18e6450 in the signature.
- A " ^ " denotes bit-wise XOR operation of same-width vectors.
- @x-y denotes bytes x through y (including both of them) of the incoming packet, where byte 0 is the first byte of the IP header. In other words, we consider all byte-offsets as offsets into a packet where the framing layer header has been stripped out. Therefore, the source IPv4 address is referred to as @12-15, while the destination v4 address is referred to as @16-19.

- @x-y, @v-w denotes concatenation of bytes x-y, followed by bytes v-w, preserving the order in which they occurred in the packet.

All hash function variations (IPv4 and IPv6) follow the same general structure. Specific details for each variation are described in the following section. The hash uses a random secret key of length 320 bits (40 bytes); the key is stored in the RSS Random Key Register (RSSRK).

The algorithm works by examining each bit of the hash input from left to right. Our nomenclature defines left and right for a byte-array as follows: Given an array K with k bytes, our nomenclature assumes that the array is laid out as follows:

- K[0] K[1] K[2] … K[k-1]

K[0] is the left-most byte, and the MSB of K[0] is the left-most bit. K[k-1] is the right-most byte, and the LSB of K[k-1] is the right-most bit.

ComputeHash(input[], N)

```
For hash-input input[] of length N bytes (8N bits) and a random secret key
K of 320 bits
Result = 0;
For each bit b in input[] {
if (b == 1) then Result ^= (left-most 32 bits of K);
shift K left 1 bit position;
}
return Result;
```

7.1.2.8.1.1 Pseudo-Code Examples

The following four pseudo-code examples are intended to help clarify exactly how the hash is to be performed in four cases, IPv4 with and without ability to parse the TCP header, and IPv6 with and without a TCP header.

Hash for IPv4 with TCP

Concatenate SourceAddress, DestinationAddress, SourcePort, DestinationPort into one single byte-array, preserving the order in which they occurred in the packet: Input[12] = @12-15, @16-19, @20-21, @22-23.

```
Result = ComputeHash(Input, 12);
```

Hash for IPv4 with UDP

Concatenate SourceAddress, DestinationAddress, SourcePort, DestinationPort into one single byte-array, preserving the order in which they occurred in the packet: Input[12] = @12-15, @16-19, @20-21, @22-23.

```
Result = ComputeHash(Input, 12);
```

Hash for IPv4 without TCP

Concatenate SourceAddress and DestinationAddress into one single byte-array

```
Input[8] = @12-15, @16-19
```

```
Result = ComputeHash(Input, 8)
```

Hash for IPv6 with TCP

Similar to above:

```
Input[36] = @8-23, @24-39, @40-41, @42-43
Result = ComputeHash(Input, 36)
```

Hash for IPv6 with UDP

Similar to above:

```
Input[36] = @8-23, @24-39, @40-41, @42-43
Result = ComputeHash(Input, 36)
```

Hash for IPv6 without TCP

```
Input[32] = @8-23, @24-39
Result = ComputeHash(Input, 32)
```

7.1.2.8.2 Redirection Table

The redirection table is a 128-entry structure, indexed by the seven LSBs of the hash function output.

System software might update the redirection table during run time. Such updates of the table are not synchronized with the arrival time of received packets. Therefore, it is not guaranteed that a table update takes effect on a specific packet boundary.

7.1.2.8.3 RSS Verification Suite

Assume that the random key byte-stream is:

```
0x6d, 0x5a, 0x56, 0xda, 0x25, 0x5b, 0x0e, 0xc2,
0x41, 0x67, 0x25, 0x3d, 0x43, 0xa3, 0x8f, 0xb0,
0xd0, 0xca, 0x2b, 0xcb, 0xae, 0x7b, 0x30, 0xb4,
0x77, 0xcb, 0x2d, 0xa3, 0x80, 0x30, 0xf2, 0x0c,
0x6a, 0x42, 0xb7, 0x3b, 0xbe, 0xac, 0x01, 0xfa
```

Table 7-9 IPv4

Destination Address/Port	Source Address/Port	IPv4 only	IPv4 with TCP
161.142.100.80 :1766	66.9.149.187 :2794	0x323e8fc2	0x51ccc178
65.69.140.83 :4739	199.92.111.2 :14230	0xd718262a	0xc626b0ea
12.22.207.184 :38024	24.19.198.95 :12898	0xd2d0a5de	0x5c2b394a
209.142.163.6 :2217	38.27.205.30 :48228	0x82989176	0xafc7327f
202.188.127.2 :1303	153.39.163.191 :44251	0x5d1809c5	0x10e828a2

The IPv6 address tuples are only for verification purposes, and may not make sense as a tuple.

Table 7-10 IPv6

Destination Address/Port	Source Address/Port	IPv6 only	IPv6 with TCP
3ffe:2501:200:3::1 (1766)	3ffe:2501:200:1fff::7 (2794)	0x2cc18cd5	0x40207d3d
ff02::1 (4739)	3ffe:501:8::260:97ff:fe40:efab (14230)	0x0f0c461c	0xdde51bbf
fe80::200:f8ff:fe21:67cf (38024)	3ffe:1900:4545:3:200:f8ff:fe21:67cf (44251)	0x4b61e985	0x02d1feef

7.1.3 MAC Layer Offloads

7.1.3.1 CRC Strip

The 82599 potentially strips the L2 CRC on incoming packets.

CRC strip is enabled by the HLREG0.RXCRCSTRP bit. When set, CRC is stripped from all received packets.

The policy for CRC strip is as follows:

- When RSC is enabled on any queue, the global CRC strip bit should be set (HLREG0.RXCRCSTRP = 1).
- When either LinkSec or IPsec are enabled, the global CRC strip bit should be set (HLREG0.RXCRCSTRP= 1b), since the payload of the packet changes and the CRC value is stale due to it.

7.1.4 Receive Data Storage in System Memory

The 82599 posts receive packets into data buffers in system memory.

The following controls are provided for the data buffers:

- The SRRCTL[n].BSIZEPACKET field defines the data buffer size. See section Section 7.1.2 for packet filtering by size.
- The SRRCTL.BSIZEHEADER field defines the size of the buffers allocated to headers (advanced descriptors only).
- Each queue is provided with a separate SRRCTL register.

Receive memory buffer addresses are word (2 x byte) aligned (both data and headers).

7.1.5 Legacy Receive Descriptor Format

A receive descriptor is a data structure that contains the receive data buffer address and fields for hardware to store packet information. Upon receipt of a packet for this device, hardware stores the packet data into the indicated buffer and writes the length, status and errors to the receive descriptor. If SRRCTL[n].DESCTYPE = zero, the 82599 uses the Legacy Rx descriptor as listed in Table 7-11. The shaded areas indicate fields that are modified by hardware upon packet reception (so-called descriptor write-back).

Legacy descriptors should not be used when advanced features are enabled: SCTP, Virtualization, DCB, LinkSec, IPSec, FCoE or RSC. Packets that match these cases might be dropped from queues that use legacy receive descriptors.

Refer to Table 7-11 and the field descriptions that follow.

Table 7-11 Legacy Receive Descriptor (RDESC) Layout

	63 48	47 40	39 32	31 16	15 0
0	Buffer Address [63:0]				
8	VLAN Tag	Errors	Status	Fragment Checksum	Length

Buffer Address (64-bit offset 0, 1st line)

Physical address in host memory of the received packet buffer.

Length Field (16-bit offset 0, 2nd line)

The length indicated in this field 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.

Fragment Checksum (16-bit offset 16, 2nd line)

This field is used to provide the fragment checksum value. This field is equal to the unadjusted 16-bit ones complement of the packet. Checksum calculation starts at the L4 layer (after the IP header) until the end of the packet excluding the CRC bytes. In order to use the fragment checksum assist to offload L4 checksum verification, software might need to back out some of the bytes in the packet. For more details see Section 7.1.13.

Status Field (8-bit offset 32, 2nd line)

Status information indicates whether the descriptor has been used and whether the referenced buffer is the last one for the packet. Error status information is listed in Table 7-13.

Table 7-12 Receive Status (RDESC.STATUS) Layout

7	6	5	4	3	2	1	0
PIF	IPCS	L4CS	UDPCS	VP	Reserved	EOP	DD

EOP (End of Packet) and DD (Descriptor Done)

Refer to the following table:

DD	EOP	Description
0	0	Software setting of the descriptor when it hands it to the hardware.
0	1	Reserved (invalid option).
1	0	A completion status indication for non-last descriptor of a packet that spans across multiple descriptors. It means that the hardware is done with the descriptor and its buffers while only the *Length* fieldis valid on this descriptor.
1	1	A completion status indication of the entire packet. Software might take ownership of its descriptors while all fields in the descriptor are valid.

VP (VLAN Packet)

The VP field indicates whether the incoming packet's type is a VLAN (802.1q). It is set if the packet type matches VLNCTRL.VET. Furthermore, if the RXDCTL.VME bit is set then active VP bit also indicates that VLAN has been stripped in the 802.1q packet. For a further description of 802.1q VLANs, see Section 7.4.

IPCS (Ipv4 Checksum), L4CS (L4 Checksum), UDPCS (UDP Checksum)

These bits are described in the following table. In I/O mode: switched packets from a local VM that do not use the Tx IP checksum offload by hardware have the IPCS equal to zero; switched packets from a local VM that do not use the Tx L4 checksum offload by hardware have the L4CS and UDPCS equal to zero.

L4CS	UDPCS	IPCS	Functionality
0	0	0	Hardware does not provide checksum offload.
0	0	1	Hardware provides IPv4 checksum offload. Pass/fail indication is provided in the *Error* field – IPE.
1	0	1 / 0	Hardware provides IPv4 checksum offload if IPCS is active along with TCP checksum offload. Pass/fail indication is provided in the *Error* field – IPE and TCPE
1	1	1 / 0	Hardware provides IPv4 checksum offload if IPCS is active along with UDP checksum offload. Pass/fail indication is provided in the *Error* field – IPE and TCPE

See Section 7.1.11 for a description of supported packet types for receive checksum offloading. IPv6 packets do not have the *IPCS* bit set, but might have the *L4CS* bit and *UDPCS* bit set if the 82599 recognizes the transport header.

PIF (Non Unicast Address)

The *PIF* bit is set on packets with a non-unicast destination Ethernet MAC address — multicast and broadcast.

Error Field (8-bit offset 40, 2nd line)

Table 7-13 and the following text describes the possible errors reported by the hardware.

Table 7-13 Receive Errors (RDESC.ERRORS) Layout

7	6	5	4	3	2	1	0
IPE	TCPE	Reserved	Reserved	Reserved	Reserved	Reserved	RXE

IPE (Ipv4 Checksum Error)

The IP checksum error is valid only when the *IPCS* bit in the *Status* field is set (indicating that the hardware validated the IP checksum). This bit is meaningful only on the last descriptor of a packet while the *EOP* bit is set as well. Packets with IP error are posted to host memory regardless of the store bad packet setting (FCTRL.SBP).

TCPE (TCP/UDP Checksum Error)

The TCP/UDP checksum error is valid only when the *L4CS* bit in the *Status* field is set (indicating that the hardware validated the L4 checksum). This bit is meaningful only on the last descriptor of a packet while the *EOP* bit is set as well. Packets with a TCP/UDP error are posted to host memory regardless of the store bad packet setting (FCTRL.SBP).