endace
accelerated

Enhanced Packet Processing Software Guide

EDM04-26

Protection Against Harmful Interference

When present on equipment this manual pertains to, the statement "This device complies with part 15 of the FCC rules" specifies the equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the Federal Communications Commission [FCC] Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

Extra Components and Materials

The product that this manual pertains to may include extra components and materials that are not essential to its basic operation, but are necessary to ensure compliance to the product standards required by the United States Federal Communications Commission, and the European EMC Directive. Modification or removal of these components and/or materials, is liable to cause non compliance to these standards, and in doing so invalidate the user's right to operate this equipment in a Class A industrial environment.

Disclaimer

Website

http://www.endace.com

Copyright 2008 Endace Technology Ltd. All rights reserved.

Contents

Introduction

This document explains how to configure the enhanced packet processing. It covers the capabilities of each of the enhanced processing steps (parse, extract, classify, colorize and steer) and describes the software interface used to configure them. Then provides programming examples to demonstrate how to combine these functions to achieve high level functionality such as filtering, Hash Load Balancing and duplication.

This document is applicable to the following DAG cards:

- 5.0SG2A
- 5.2SXA (5.2SAA and 5.2XAA)
- 5.4GA
- 5.4SA-12
- 5.4SGA-48

For information on accelerated DAG cards with daughter board Co-Processors see *EDM02-02 Co-Processor IP Filtering Software Manual*.

Default packet processing functions

When performing packet capture all Endace DAG cards perform the same basic functions:

1. Receive traffic on one or more physical ports.
2. De-encapsulate and extract packets from the received bit stream. Ethernet or Packet Over Sonet (POS) encapsulations are typical, depending on port type.
3. Wrap each received packet in an Extended Record Format (ERF) record. Insert a timestamp into the ERF header to record the exact time the packet was received. Also insert the port number the packet was received on, and set the ERF record type to *ERF 1. TYPE_HDLC_POS* or *ERF 2. TYPE_ETH*. For multi-port cards, merge the traffic from all ports into a single incoming packet stream, ensuring timestamp order per port is maintained.
4. Write the ERF record via hardware controlled direct memory access (DMA) to a stream buffer that is shared with the application and resides on the host computer.

Steps 1-4 implement 100% capture of all packets and are documented in each DAG Card User Guide.

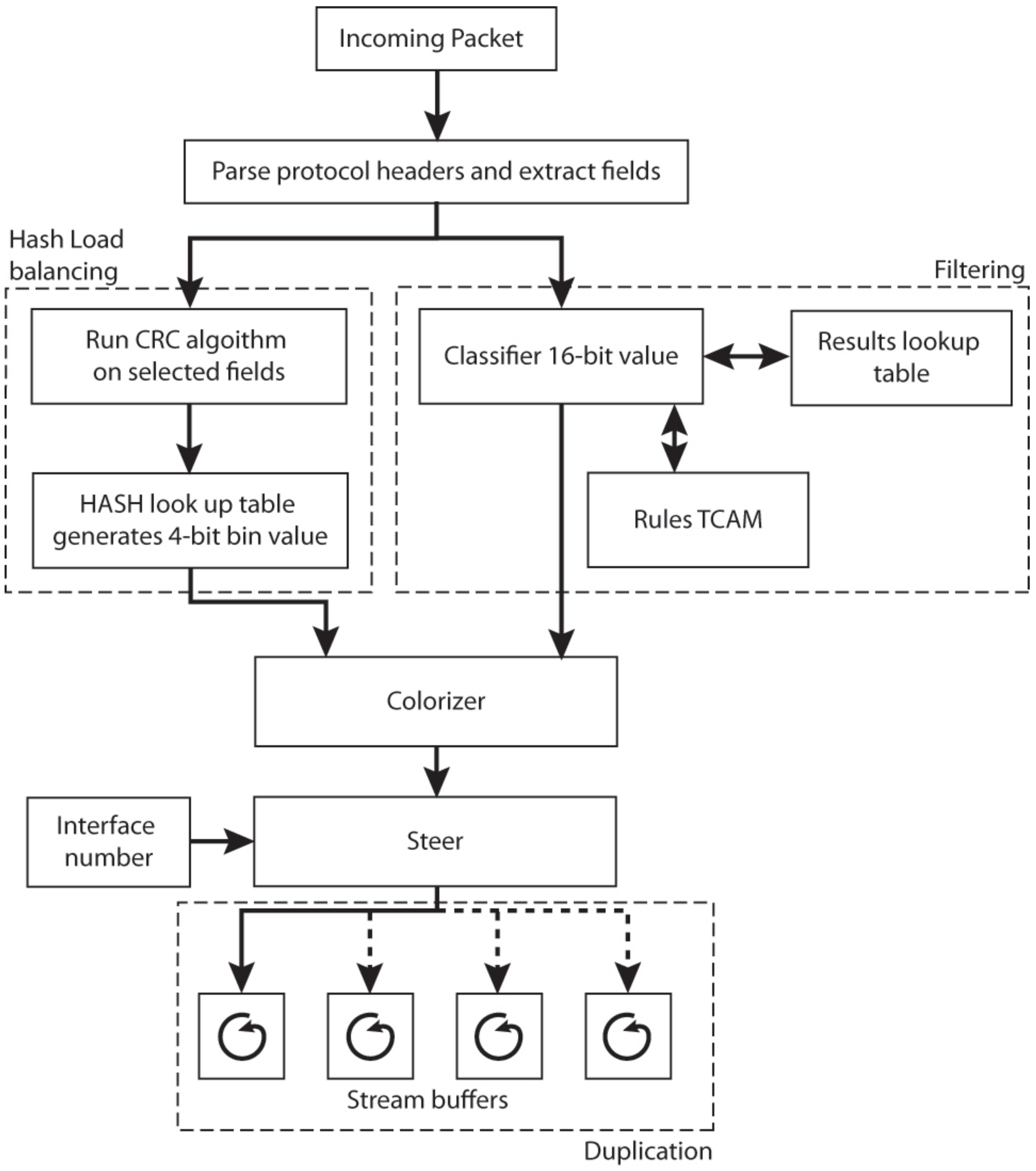
Enhanced packet processing

Some DAG cards and their associated software provide enhanced packet processing functions that can be used to simplify and accelerate packet processing software on the server. You may need to download the latest software and firmware release from the Endace website to access these features on your DAG card. Refer to the most recent DAG Card User Guide to determine if Enhanced packet processing is available on your DAG card.

With the enhanced packet processing capability the following types of enhanced capabilities can be achieved:

- **Filtering:** The goal is to save bus bandwidth and reduce the amount of traffic the application must process by filtering out unneeded traffic before it is written into the application's stream buffer. Filtering requires identifying the desired fields in the packet's protocol headers, performing a comparison function against these fields to identify (i.e. classify) traffic streams, and dropping the unneeded traffic while letting the desired streams pass through to the stream buffer.
- **Hash Load Balancing (HLB):** The goal of Hash Load Balancing is to receive a mixed stream of packets and spread the traffic evenly across several stream buffers. This is typically used to optimize the parallel processing done by multi-core servers since each core will only see a subset of the full traffic load. You normally want to distributing traffic equally (or roughly equally) across the stream buffers while preserving "flow coherence", meaning, all packets associated with a single TCP/IP session must always go to the same stream buffer. Hashing is the tool used to classify the flow coherent streams and mathematically assign a "bin number" to each packet. Bin numbers are then associated with specific stream buffers to achieve the desired load balancing. A full discussion of how hash functions work is beyond the scope of this document, but in essence the Hash Load Balancing performs packet classification and steering similar to filtering and steering.
- **Classify and Colorize:** The goal is to have the hardware perform the first level of traffic classification and then write the classification result (commonly called the packet's colour) into a field in the extended ERF record. Later, when processing the packet, the application can accelerate its processing by doing a simple lookup on the packet's colour and jump directly to the appropriate processing function.
- **Classify and Steer:** Similar to filtering, but the goal is not to eliminate (i.e. drop) traffic but rather spread traffic across multiple stream buffers in order to take better advantage of multi-core processors. For example, all traffic to and from a specific IP address could be sent to one stream buffer, while all other traffic sent to another.
- **Duplication:** For this document, packet duplication is defined as writing a single packet into more than one stream buffer. It does not mean writing the packet twice into the same stream buffer. Packet duplication can simplify software architectures where different types of processing needs to be done on the same packet. For example, all traffic can be sent to stream buffer 0 for use by a capture to disk applications, while only a subset of traffic is sent to stream buffer 2 for handling by a traffic analysis application.

When properly configured by the application software, the following processing steps work together to implement the enhanced packet processing functions. Refer to the following diagram for a graphical representation of these steps.


Incoming Packet
Parse protocol headers and extract fields
Hash Load balancing
Run CRC algoithm on selected fields
HASH look up table generates 4-bit bin value
Filtering
Classifier 16-bit value
Results lookup table
Rules TCAM
Colorizer
Interface number
Steer
Stream buffers
Duplication

Enhanced packet processing functions

1. Receive traffic on one or more physical ports. For further details, see Packet Receive (page 5).
2. De-encapsulate and extract packets from the received bit stream. Ethernet or Packet Over SONET (POS) encapsulations are assumed, depending on port type.
3. Wrap each received packet in an Extended Record Format (ERF) record. Insert the timestamp, port number and set the ERF record type to *ERF 1. TYPE_HDLC_POS* or *ERF 2. TYPE_ETH*. For multi-port cards, merge the traffic from all ports into a single incoming packet stream.
4. Extract the EtherType field or PPP protocol field, and a subset of the TCP/IP packet header fields. These are called the *extracted fields* and are used by the packet classification function below. For further details, see Packet Parsing and Header Field Extraction (page 6).
5. Classify the packet using the extracted fields. Two types of classification can be performed simultaneously: Hash Load Balancing and field matching.
   - Hash Load Balancing – runs a configurable subset of the extracted fields through a standard CRC32 algorithm. The output from the algorithm is 10 bits, which are given to a configurable look up table to yield 4 bits that identify which of 16 "hash bins" the packet belongs to. For further details, see Classify – Hash Load Balancing (page 8).
   - Field matching (also called filtering) – using user defined Tertiary Content Addressable Memory (TCAM), simultaneously compares each bit of the packet's extracted fields against a number of bit masks (comprised of 0/1/don't care). It then returns the location of the highest priority (lowest numbered) memory location that matches the extracted fields. This location is then used to address into a colour lookup table, which returns the 16-bit value that has been defined by the application for that memory location. For further details, see Classify – Filtering (page 10).
6. Merge the hash result with the filtering return value to define the packet's "color". The packet's color is the key output of the packet classification process. The packet's colour is inserted in the extended header of the ERF record. It is often used by application software to accelerate packet processing since the packet classification process has been performed efficiently in hardware. Note that only a subset of the colour value is used to guide packet steering below. For further details, see Colorize (page 11).
7. Using the packet's color (and optionally the port ID) index into a configurable lookup table and determine whether to DMA the packet to one or more stream buffers in the host, or drop (i.e. filter out) the packet. The number of stream buffers supported by each DAG card depends on type of card. Sending a packet to more than one stream buffer is valid and results packet duplication to multiple stream buffers. For further details, see Steering (page 12).

Packet Receive

Functional description

Steps 1-3 above describe the Packet Receive function. Traffic is received on one or more physical ports, packets are de-encapsulate and extracted from the received bit stream, each packet is wrapped in an Extended Record Format (ERF) record and the time-stamp and receive port are written into the ERF header. Traffic from all ports is merged into a single datapath.

How to program

Configuration of the Packet Receive function is fully described in your DAG Card User Guide, which describes how to load and initialize the DAG card and prepare the packet receive function for basic packet capture. For the remainder of this document, it is assumed that:

- the DAG card drivers have been installed,
- the appropriate firmware has been loaded on the card, and,
- for DAG cards that support multiple protocols, the DAG card has been configured to receive the desired protocol.

Packet Parsing and Header Field Extraction

After the packet receive process is complete, the DAG card extracts a subset of the packet's headers for later use by the Hash Load Balancing and classification functions. This extraction process differs slightly depending on whether the interface is Ethernet or SONET/SDH.

Functional description - Ethernet

The TCP/IP header's location is determined dynamically (i.e. on a packet by packet basis) because the TCP/IP header's offset from the beginning of the packet depends on the layer 2 encapsulation used, as defined by the EtherType field. Supported IP encapsulation options allow 0, 1, or 2 VLAN tags. Although the VLAN tags cannot be used in the hashing or filtering steps, the presence of VLAN tags does not interfere with IP and TCP/UDP header field extraction. VLAN tags are simply skipped over.

Header field extraction proceeds until the protocol parser has extracted all the desired fields or until the parser can proceed no further because the protocol is not supported. The parser assigns zeros to any fields it could not extract from the packet. The following are the maximum number of fields that can be extracted:

EtherType or PPP protocol	(16 bits)
IPv4 Source IP Address	(32 bits)
IPv4 Destination IP Address	(32 bits)
IP Protocol field	(8 bits)
Source TCP or UDP port	(16 bits)
Destination TCP or UDP port	(16 bits)
TCP Flags field	(6 bits)
Receive port ID	(1 bit)

For Ethernet, the parsing of these fields proceeds as follows:

1. Set the receive port ID based on which port the packet arrived on, and initialize all extracted fields to zero.
2. Look at the 16 bit EtherType value. If the EtherType value is VLAN 802.1Q (0x8100) then skip over theVLAN tag, and look at the EtherType value just after the VLAN tag. If the EtherType is again 0x8100 then skip over the second VLAN tag.
3. After VLAN tag skipping, extract the EtherType field. If the EtherType is not IP (0x0800) then stop parsing. Note: for the application to classify these these types of packets the filter rule should only look at receive port and EtherType, all other fields in the filter rule should be set to "don't care."

   (note: if the parser gets to this step then we know the extracted EtherType value is IP = 0x0800) Extract the IP source address, the IP destination address, and the IP Protocol field.
4. Examine the IP protocol field. If it is neither TCP (6) nor UDP (17) then stop parsing. In this case filtering can be performed on receive port, EtherType, IP protocol and IP src/dst. All other fields in the filter rule should be set to "don't care."
5. If the IP protocol field is UDP then extract the UDP source and destination addresses and then stop parsing. Filtering can be performed on receive port, EtherType, IP Protocol, IP src/dst, and UDP src/dst. The TCP flags field should be set to "don't care."
6. If the IP protocol field is TCP then extract the TCP source and destination addresses, and the lower 6 bits from the TCP flags field (also called the TCP control field). The bits extracted are URG, ACK, PSH, RST, SYN, and FIN. Then stop parsing. All fields are available for filtering.

The extracted fields are concatenated together and form a contiguous 127-bit word that is passed to the TCAM for filter lookup.

Functional description – SONET/SDH

For SONET/SDH ports the encapsulations supported are PPP in HDLC like framing (RFC1661) and Cisco HDLC (POS in this document).

The parsing proceeds as follows:

1. Set the receive port ID based on which port the packet arrived on, and initialize all extracted fields to zero.
2. Examine the 16 bit HDLC protocol field. If the value is POS (0x0F00) or PPP (0xFF03) then the 16 bits after the protocol field are extracted as the EtherType or the PPP protocol field respectively.
3. For all other HDLC protocol field values, stop parsing. In this case filtering can only be performed on the receive port bit. All other fields should be set to "don't care."
4. If POS and the extracted EtherType value is not IP (0x0800) then stop parsing. If PPP and the protocol type is not IP (0x0021) then stop parsing. In either case filtering can only be performed on receive port and EtherType/PPP protocol. All other fields should be set to "don't care."
5. If POS and the extracted EtherType value is IP, or if PPP and the protocol type is IP, then extract the IP source address, the IP destination address, and the IP Protocol field.
6. Examine the IP protocol field. If it is not TCP (6) or UDP (17) then stop parsing. In this case filtering can be performed on receive port, EtherType, and IP src/dst. All other fields should be set to "don't care."
7. If the IP protocol field is UDP then extract the UDP source and destination addresses and then stop parsing. Filtering can be performed on receive port, EtherType, IP src/dst, and UDP src/dst. The TCP flags field should be set to "don't care."
8. If the IP protocol field is TCP then extract the TCP source and destination addresses, and the lower 6 bits from the TCP flags field (also called the TCP control field). The bits extracted are URG, ACK, PSH, RST, SYN, and FIN. Then stop parsing. All fields are available for filtering.

The extracted fields are concatenated together and form a contiguous 127-bit word that is passed to the TCAM for filter lookup.

How to configure

There is no configuration required for the packet parsing function. As described above is the default behaviour.

Classify – Hash Load Balancing

Functional description

Hash Load Balancing occurs on a configurable subset of the extracted header fields described above. The same subset applies to all packets received on all ports. Each extracted field is padded out to 32 bits, XOR'd together to form a single 32 bit word, which is then passed through a standard CRC32 algorithm. The 10 bit output of the CRC algorithm is sent to a user configurable lookup table called the Hash Association Table (HAT). The output of the HAT is a 4 bit value, which is known as the packet's hash-bin.

Hash Load Balancing in this way maintains flow coherence, meaning, all packets from a single TCP/IP session produce the same 10-bit hash result so that the bi-directional stream of packets forming the TCP/IP session are always kept together for processing by a single application. Hash Load Balancing is, in some ways, the inverse of filter-based classifying. Filters are typically used to classify packets by identifying specific TCP/IP flows. With Hash Load Balancing, the TCP/IP headers are used to generate what is essentially a random value (the 10 bit hash result). It is up to the application to determine how to sort these randomized flows into 16 groups (the 16 hash bins). Typically the goal is load balancing – the process of assign a well defined subset of flows to each core of a multi-core server.

For example, if there are 4 processor cores available to handle packet flows, the application could configure the HAT to sort all traffic into four bins, leaving the remaining 8 bins empty. Those four bins would then be steered to four stream buffers (see Steering (page 12)), and each processor core would be given a single buffer to handle. Alternatively, the HAT could be configured to use all 16 bins, with steering used to assign four bins per stream buffer.

The obvious question is why use a HAT at all? Why not just produce 4 bits directly from the CRC32 algorithm? The purpose of the HAT is to allow applications to "tweak" the assignment of flows to bins. For example, if the application finds that one flow is generating 25% of the traffic, it could use the HAT to map the 10-bit hash result for that flow to its own hash bin.

Note: The HAT can be modified "on-the-fly" while packet capture is occurring. However, changing the HAT during packet capture may result in a temporary splitting of a single TCP/IP flow across two bins.

As mentioned above, a subset of the extracted header fields can be used for hashing. The subset to use must be configured before packet capture is enabled, and it cannot be changed without halting packet capture. The subsets supported are as follows:

- 2-tuple (IP src + IP dst)
- 3-tuple (2-tuple + IP Protocol)
- 4-tuple (3-tuple + Interface)
- 5-tuple (3-tuple + TCP/UDP src + TCP/UDP dst)
- 6-tuple (5-tuple + Interface)

The DAG card's Hash Load Balancing function is optional, and is enabled using the `dagconfig` tool provided by Endace. Hash Load Balancing can be turned on and off without reprogramming the HAT (see below) but data capture must be halted in order to disable or enable Hash Load Balancing. If Hash Load Balancing is enabled, the ERF Type of captured packets is changed to *ERF 19. TYPE_COLOR_HASH_POS* record with Hash Load Balancing or *ERF 20. TYPE_COLOR_HASH_ETH* variable length record with Hash Load Balancing. See *EDM11-01 ERF types* for details. The 4-bit hash result can be used to define the lower 4 bits of the packet's colour field in the ERF header.

How to program

Enable hashing:

```
dagconfig -d0  -S hash_encoding_from_ipf=on
```

Turn off the backward compatible use of filter result (see Colorize (page 11)):

```
dagconfig -d0  -S shift_colour=on
```

Configure the fields to be used for CRC32 algorithm by specifying the number of tuples to use (see definition of tuples on previous page). In this example, the 4-tuple fields are used:

```
dagconfig -d0  -S n_tuple_select=4
```

Load the HAT to map the 10-bit hash value to the 4-bit bin value. The syntax is to specify the percent range of hash values to assign to each bin, specified as a range of integers (0-999). The following example evenly spreads the hash values across the lower 4 bins, leaving the upper 12 bins empty:

```
dagconfig -d0 -S hat_range=0-249:250-499:500-750:750:999
```

The next example spreads the hash values unevenly across bins 1,4, 10, and 12 , leaving the rest empty. There is no practical application for this example, but it shows the syntax.

```
dagconfig -d0 -S hat_range= :0-1: : : 2-499 : : : : : : 500-800 :  :801-999 :
: :
```

dagconfig attributes - HAT

The following `dagconfig` attributes are applicable to the HAT (Hash Association Table) part of Enhanced Packet Processing. Using the `dagconfig` -S and -G options you can set and get values of the listed attributes.

Option	Description
`hash_encoding_from_ipf`	Enables (on) or disables (off) hash encoding in the DAG card.
`n_tuple_select`	Sets the hashing tuple value (2, 3, 4, 5 or 6).
`hat_range`	Sets the HAT ranges for the DAG card. This determines how the hash output are distributed over to the 16 banks. The ranges is 0 to 1000. **Note:** You may need to tune these ranges to ensure even distribution. For example : `dagconfig -d0 -S hat_range=0-90:90-160:160-200:200-250:250-310:310-370:370-430:430-500:500-560:560-620:620-680:680-750:750-810:810-870:870-940:940-1000`
`shift_colour`	Enables (on) the higher 2 bits of color for the hash value. If not set, the user tag's lower two bits are overwritten by the hash value.

For more details about `dagconfig`, see you DAG Card User Guide.

Classify – Filtering

Functional description

After the packet header fields are extracted the next step (optionally done in parallel with Hash Load Balancing below) is to classify the packet via protocol header matching (also called filtering).

A user configurable Ternary Content Addressable Memory (TCAM) is used to simultaneously compare each bit of the packet's extracted fields against a series of bit masks (comprised of 0/1/don't care) that have been loaded into the TCAM by the application. These bit masks are typically called the "filter rules" against which the packet header is compared. The size of the TCAM determines the number of rules that can be stored by the TCAM. The size is product specific and is documented in your DAG Card User Guide.

When classifying each packet, the extracted header fields are presented to the TCAM, which simultaneously compares the extracted fields against each rule in the TCAM. Depending on the filter rules loaded into the TCAM, more than one match is possible per packet. No indication of multiple matches is provided – only the lowest numbered address that matches is returned. If no match is found the TCAM returns the highest address supported by that TCAM. Typically a "match everything" rule is included at the end of the filter record file loaded into the TCAM in order to ensure a rule match always occurs at a configured location.

Associated with the TCAM is a standard memory the same size as the TCAM. The match location returned by the TCAM is used as the address for reading this memory, which contains 16 bit values. This memory must be initialized by the application in order to associate each rule with the desired 16-bit classification result. The classification result is used, optionally with the Hash Load Balancing result, to determine the packet's colour as described below.

How to program

The classifier TCAM and associated memory are enabled and programmed via the `filter_loader` tool provided by Endace as part of the standard DAG software. Please refer to the *EDM04-28 filter_loader Software Guide* for details on how the use `filter_loader`. Filters are normally defined in a filter file, which consists of a series of (rule number, filter-value) pairs. The rule number is the classification result that will be returned if this rule is matched (i.e. the rule number sets the packet colour). More than one filter can be associated with the same rule number, so any number of filter rules can produce the same colour.

Filtering implementations on older Endace cards are slightly different than described here. Cards with Enhance Packet Processing default to a "backwards compatibility mode" that allows the filter rule files from older Endace cards to be used with the newer Endace cards. Because of this, when using the filter_loader tool, the lower two bits of the filter colour are denoted as RED and BLUE. In older DAG cards these bits directly determined how packets are steered or dropped. Setting the colour's RED bit steered the packet to stream buffer 0. Setting the BLUE bit steered the packet to stream buffer 2. Setting neither caused the packet to be dropped at the classification stage, and setting both caused the packet to be duplicated to both stream buffers. By retaining this functionality in cards with enhanced packet processing as being discussed here, existing filter rule files work as expected.

Note: The enhanced packet classifier documented here retains backward compatibility with older cards by dropping packets immediately if neither the RED nor the BLUE bit is set during classification. This should be taken into account when setting up the filter rules using `filter_loader`.

Colorize

Functional description

A packet's colour is a numeric value corresponding to the overall classification result. To colorizing a packet is simply to record this numeric value into the packet's ERF record for later use by the application. The DAG card colorizes each packet in one of two ways depending on whether hash load balancing has been enabled:

If hashing has not been enabled the ERF record type is set to *ERF 10. TYPE_COLOR_HDLC_POS* or *ERF 11. TYPE_COLOR_ETH*. The bits in the colour field of the ERF record are set as follows:

bits [0]	RED bit as set by the filter_loader
bits [1]	BLUE bit as set by the filter_loader
bits [15:2]	Bits [13:0] of the filter-rule number (also called user tag) of the matching classification rule as defined by the filter file loaded by filter_loader.

If hashing is enabled, the ERF record type is set to *ERF 19. TYPE_COLOR_HASH_POS* or *ERF 20. TYPE_COLOR_HASH_ETH*. See the *EDM11-01 ERF types* for details.

When hashing is enabled and shift_colour is off (default) the ERF record colour field is set as follows:

bits [3:0]	Hash result
bits [15:4]	Bits [13:2] of the filter-rule number (also called user tag) of the matching classification rule as defined by the filter file loaded by filter_loader.

Note that this truncates the lower 2 bits of the filter-rule number and makes filter rule configuration unnecessarily confusing when backward compatibility is not required. It is recommended that the colorizer be configured to use the lower bits of the filter-rule number. The command to achieve this is:

```
dagconfig -d0  -S shift_colour=on
```

When colour shift is on and hashing enabled the packet's colour is set as follows:

Colour bits [3:0]	Hash result
Colour bits [15:4]	Bits [11:0] of the filter-rule number.

How to program

The DAG card changes the ERF record type and colorizes each packet whenever filtering or hashing is enabled. As discussed above, using the following command is recommended when hashing is enabled:

```
dagconfig -d0  -S shift_colour=on
```

There is no option to turn packet colorization off, so the packet colour is always written into the ERF header if either hashing or filtering is enabled.

dagconfig attributes - CAT

The following `dagconfig` attributes are applicable to the CAT (Color Association Table) part of Enhanced packet processing. Using the `dagconfig` -S and -G options you can set and get values of the listed attributes.

Option	Description
`bank_select`	Activates a CAT bank without loading a new configuration.
`by_pass`	Enables (off) or disables (on) the CAT.
`interface_overwrite_enable`	Enables (on) or disables (off) the use of Port ID.
`deliberate_drop_count`	Read only. Use with `dagconfig -G` only. Displays the number of packets dropped at the CAT.

For more details about `dagconfig`, see you DAG Card User Guide.

Steering

Functional description

The DAG card hardware performs packet steering per-packet, per stream-buffer. Each stream buffer's steering logic uses a subset of the packet's colour (and optionally the port ID) to address into a lookup table and determine whether to DMA the packet to the stream buffer, or to drop (i.e. filter out) the packet. Each stream buffer functions independently of the others, but caution should be used when allowing packets to be duplicated across multiple stream buffer since this can drastically increase the bus bandwidth and cause packet loss. The number of stream buffers supported by each DAG card depends on the type of card.

The internal mechanism that matches a packet's colour to the action to be taken by each stream buffer manager is determined by a lookup table called the Colour Association Table, or CAT. The CAT is addressed using a 14-bit address created in one of two ways depending on whether use of the port ID has been enabled:

port ID disabled:	Bits [13:0] = Bits [13:0] from the packet's colour
port ID enabled:	Bits [13:12] = The interface number the packet was captured on Bits [11:0] = Bits [11:0] from the packet's colour

Each of the 16K rows in the CAT has a bit corresponding to each stream buffer supported by the DAG card. The number of stream buffers is DAG card specific, and is specified in the DAG Card User Guide. When the bit corresponding to the stream buffer is set to '1' the packet is steered to that stream. Packet duplication occurs when more than one bit is set, meaning, the same packet is written to more than one stream buffer. If all the bits are set to '0' the packet will be dropped.

For example, assume a packet arrives for steering with its colour set to 128. For a 4 stream buffer DAG card, the following examples describe the steering result based on different CAT table values:

Location 128 = 0000	drop the packet
Location 128 = 0001	copy the packet to stream buffer 0
Location 128 = 0101	copy the packet to stream buffers 0 and 4 (note: receive buffers are numbered 0, 2, 4,...)

The CAT is arranged in two banks – active and standby. To change the CAT modify the standby bank and then flip active and standby.

Each stream buffer has an optional snap-length. The packet stored in the ERF record will be shortened (snapped) to the snap length. The snap length must be a multiple of 8 bytes. The actual packet length is recorded in the ERF record header, in the wlen field. If the packet is shorter than the snap length, the ERF record is lengthened (using padding) to the nearest 8-byte boundary. The length field in the ERF record records the actual packet size.

The steering function maintains a counter of all packet that are intentionally dropped because no stream buffer is configured to receive it. Under specific conditions the controller for each stream buffer may need to drop packets that are intended to be captured. This only occurs in two scenarios:

1. The application is not keeping up, the stream buffer is full, and the on-card FIFO used for that stream buffer has overflowed.
2. There is, in aggregate, more traffic being received than can be sent over the computer bus due to the inherent bandwidth limit of the bus. Performing a lot of packet duplication can create this scenario. If the bus becomes too busy then typically all stream buffers begin to experience some packet loss until the bus bandwidth is free again.

Note: If neither hashing nor filter based classification are enabled all traffic is steered to stream buffer 0. To maintain backwards compatibility, for these ERF record types (ERF type is 1 or 2) the colour field within the ERF header is used as a drop counter field. The steering hardware updates the drop counter field on every packet to reflect the number of dropped packets since the last packet sent.

How to program

The following examples demonstrate how to initialize and configure the steering function.

This example displays which CAT bank is currently active.

```
dagconfig -d0  -G bank_select
```

This example activates a bank without loading new configuration (0/1 are valid).

```
dagconfig -d0  -S bank_select=1
```

This example disables the CAT.

```
dagconfig -d0 -S by_pass=on
```

This example enables the CAT

```
dagconfig -d0 -S by_pass=off
```

This example enables use of port ID

```
dagconfig -d0 -S interface_overwrite_enable=on
```

This example disables use of port ID

```
dagconfig -d0 -S interface_overwrite_enable=off
```

Tip: For further details, see `dagconfig` attributes - CAT (page 11).

Configure the CAT by loading a user defined rules file

```
dagcat-setup -d1 -f cat_1.rule
```

Configure hash load balancing evenly across eight stream buffers :

```
dagcat-setup -d1 -m z8
```

A rule file consists of one-line rules that define the mapping between colour/interface/hash-bin and the stream buffers. The following table shows the keywords used to define the rules. Refer to *EDM04-27 dagcat-setup Software Guide* for a full description.

dagcat-setup filter keywords

keywords	Description
`(not)color [<from>-<to>]`	Specifies the *color range* for the rule. The use of "not" would mean that any values outside the specified range
`(not)color <colour_value>`	Specifies the *color value* for the rule (only single value). The use of "not" means that any value other than the given value.
`(not)hash [<from>-<to>]`	Specifies the *hash range* for the rule. The use of "not" means that any values outside the specified range
`(not)hash <hash_value>`	Specifies the *hash_value* for the rule (only single value). The use of "not" means that any value other than the given value.
`(not)iface [<from>-<to>]`	Specifies the *iface* range for the rule. The use of "not" means that any values outside the specified range
`(not)iface <iface_value>`	Specify the iface for the rule. (only single value). The use of "not" means that any value other than the given value.
`stream <list>`	Specify the *destination streams* for the rule. Specified as a comma separated list, for example: stream 0,2,4

Example 1:

You want to load balance all non-HTTP traffic across 4 stream buffers, and drop all HTTP traffic (TCP port 0x80).

The classifier will be set (using RED/BLUE) to drop all packets with destination TCP port of 0x80. The rest of the packets will be captured. Hashing will be enabled and 4 capture bins, each carrying 25% of the traffic, will be defined and mapped to the four stream buffers.

The rule file for the filter_loader is called reject_port80.rule and contains the following lines:

```
100 reject tcp dst-port {0000000010000000}
200 accept red all
```

Load the filter rules using filter_loader

```
dagconfig -d0  -S shift_colour=on
filter_loader -d0 --initialize --drop  --iface 0 -l ethernet  -m colour -i
reject_port80.rule
```

Enable hashing

```
 dagconfig -d0  -S hash_encoding_from_ipf=on
```

Specify the number of fields to be used for CRC

```
dagconfig -d0  -S n_tuple_select=2
```

Configure the traffic in 4 hash bins (25% each)

```
dagconfig -d0 -S hat_range=0-249 : 250-499 : 500-749 : 750-999
```

The CAT will be configured so packets marked as hash bin 0 will be set to Stream 0, packets with hash bin 1 will be set to Stream 2 and so on.

The Rule file for the `dagcat-setup` is called cat_config.rule and contains the following lines

```
hash 0 stream 0
hash 1 stream 2
hash 2 stream 4
hash 3 stream 6
```

Load the rule to the CAT

```
dagcat-setup -d0 -f cat_config.rule
```

Example 2:

You want to run 3 different applications and they all must receive the same data.

Classification should be enabled to pass all the packets. Hashing need not be enabled since all the packets are routed to the same stream(s). All CAT entries will have bits 0, 1 and 2 set, causing the packets to be duplicated to streams 0, 2 and 4.

Enable pass-all filter

```
0 accept red all src-ip
```

The Rule file for the `dagcat-setup` is called cat_config.rule and contains the following line to route all the packets to 3 streams.

```
stream 0,2, 4
```

Load the rule to the CAT

```
dagcat-setup -d0 -f cat_config.rule
```

Example 3:

You want to run 2 applications. Assume application #1 wants to receive all traffic (on stream 0), while application #2 wants to receive just the header portion of all traffic to/from IP address 192:100:100:* (on stream 2). The snap length for application #2 will be set to 128 to shorten the packets it receives.

Set the snap length on Stream 2 to 128 Bytes, the rest of the streams will have the default snap-length of 10240 Bytes. Stream 2 means 1st receive stream, so use stream_snaplength1.

```
dagconfig -d0 -S stream_snaplength1=128
```

The rule file for the filter_loader is called two_app_filter.rule and contains the following lines, which classifies the packets with the given IP address with color 100 or 101.

```
100 accept red tcp src-ip {110000001100100110010 0--------}
101 accept red tcp dst-ip {110000001100100110010 0--------}
200 accept red all
```

Load the filter rules using filter_loader

```
filter_loader -d0 --initialize --drop --iface 0 -l ethernet -m colour -i
two_app_filter.rule
```

The Rule file for the `dagcat-setup` is called cat_config.rule and contains the following lines. Any packet with color 100 or 101 should be directed to stream 0 and stream 2 and the other packets (color 200) should go to stream 0 only.

```
color [100-101] stream 0,2
color 200 stream 0
```

Load the rule to the CAT

```
dagcat-setup -d0 -f cat_config.rule
```

Version History

Version	Date	Reason
1	November 2008	First release.