MUSIC SEMICONDUCTORS

Epoch Software Manual

Revision 1

December 1, 2000

http://www.music-ic.com

email: info@music-ic.com

Contents

Overview

INTRODUCTION

This document describes the host processor software and data structures required to use the Epoch chip set in a Layer 3/Layer 4 switch application. It was designed to be read sequentially by section with appendices at hand for quick reference. The *Epoch Data Sheet* should be used in conjunction with this document. See the *Related Material* section for additional MUSIC Semiconductors documentation.

The MUSIC Semiconductors Epoch chip is a Layer 3 and Layer 4 packet switch. It can route IPv4, IP multicast, and IPX packets at wire speed using Layer 3 headers. In addition, it can classify and prioritize packets as microflows, using their Layer 3 and Layer 4 headers, or as aggregates using the Differentiated Services fields found in Layer 3 headers. Very flexible packet handling options are available to easily implement switch and router applications with Quality of Service features.

SAMPLE APPLICATION

Figure 1-1 shows a example of a WAN-edge switch using Epoch, just one of many applications for the Epoch chip set. It is an important configuration since several Epoch features are ideal for this application. Figure 1-1 is used throughout this document to explain various features of Epoch. The example shown has an internal host processor (L2port = 0) that handles packets switched to it by the Epoch chip. A separate system block (L2port = 2) is shown that handles all activities related to firewalls. The configuration has two WAN ports, one for accessing the Internet (L2port = 10) and another for accessing an Intranet (L2port = 15).

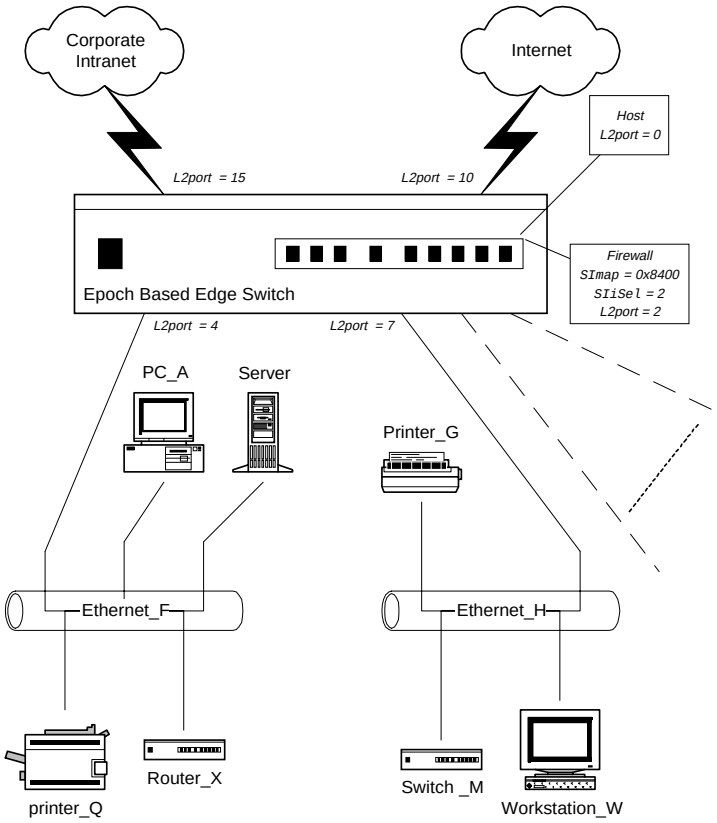


Figure 1-1: Epoch in a WAN Edge Application

The local network consists of two Ethernet subnets: Ethernet_F (L2port = 4) and Ethernet_H (L2port = 7). This configuration has network paths for nodes on the two subnets to communicate with each other and with corporate Intranet nodes as well as Internet nodes. The nodes on the subnets include workstations, PC's, printers, servers, routers and switches. The router and switch are connected to other networks. The traffic in this network can include IP packets, IPX packets and other legacy protocols such as DECnet and Appletalk. These packets can be carrying low priority traffic such as file transfers from Intranet sites, web pages from Internet sites or LAN print services; they can also carry medium priority traffic from file services such as NFS or Windows NT, or high priority traffic such as video-conferencing over the LAN or Voice over IP over the WAN.

HARDWARE ARCHITECTURE

External Elements

The key external elements of a complete Epoch application are below and in Figure 1-2.

- **Arbiter**–This element determines which L2port is active during each TDM Bus cycle.
- **Layer 2 Interface(s)–**It transmits and receives packets between the Epoch TDM Bus and Layer 2 specific devices.
- **Packet Data and Control Information Storage**–Packets are stored in the form of linked lists of 64 byte buffers.
- **RCP and Layer 3/Layer 4 Database**–Epoch uses this subsystem to retrieve Layer 3 routing information as well as Layer 4 Microflow or Differentiated Services information about a packet.
- **Queue Pointer and Packet Pointer Storage**–On chip queues and associated packet pointers external storage are used to implement prioritization and scheduling. Multicasting is achieved by assigning packets to multiple output queues.
- **Host Processor TDM Interface**–This interface is between the host processor and the Epoch TDM Bus at L2port 0. Packets requiring special processing by the host transit through it.
- **Host Processor Interface**–This interface connects the host processor to the Epoch generic processor port used to access Epoch registers, RCP registers, and all storage.
- **Host Processor**–The Host Processor initializes and configures Epoch, RCP, and associated data structures, which it also maintains during normal operations. It is also responsible for processing packets, that Epoch can not process for one reason or another, as is the case with non-IP or IPX packets, or administrative packets destined to the router itself. Several host processors can be used to divide the load or functions. This port is used for initialization and during normal operation. There is also an optional hardware interrupt feature that can be used to notify the host of important Epoch conditions. There are 14 maskable conditions that can activate the interrupt pin.

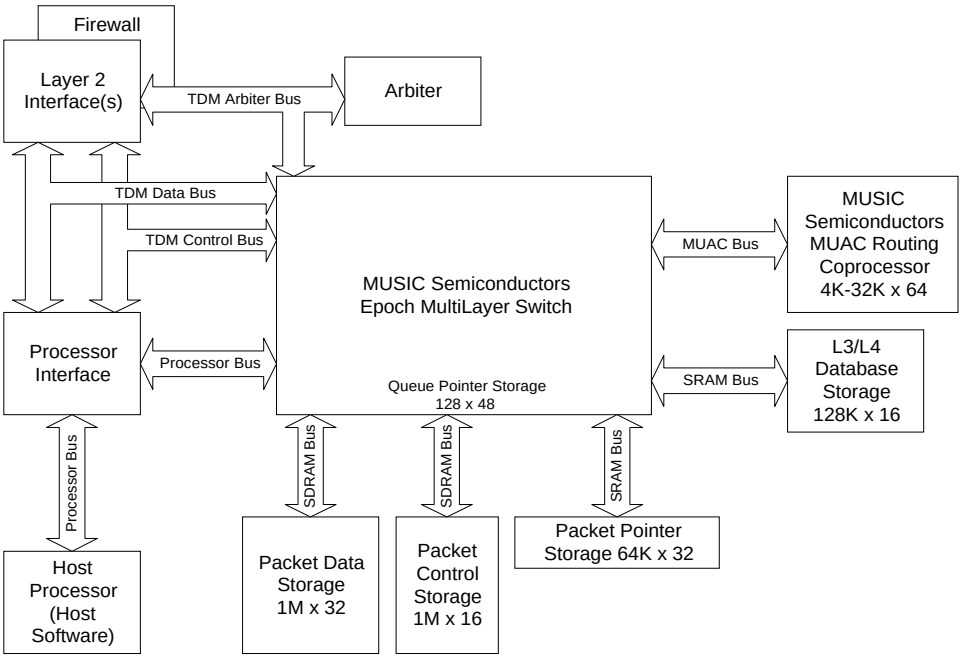


Figure 1-2: Epoch Application Major Elements

Layer 2 Interfaces and TDM Bus

All packet data enters and exits Epoch through the TDM Bus. A typical Epoch application with a focus on the TDM Bus interface is presented in Figure 1-3. This diagram shows the host processor(s) connections to the Layer 2 interfaces, the Arbiter logic and the Epoch chip. The software that resides on the host processor complex is examined later in this section.

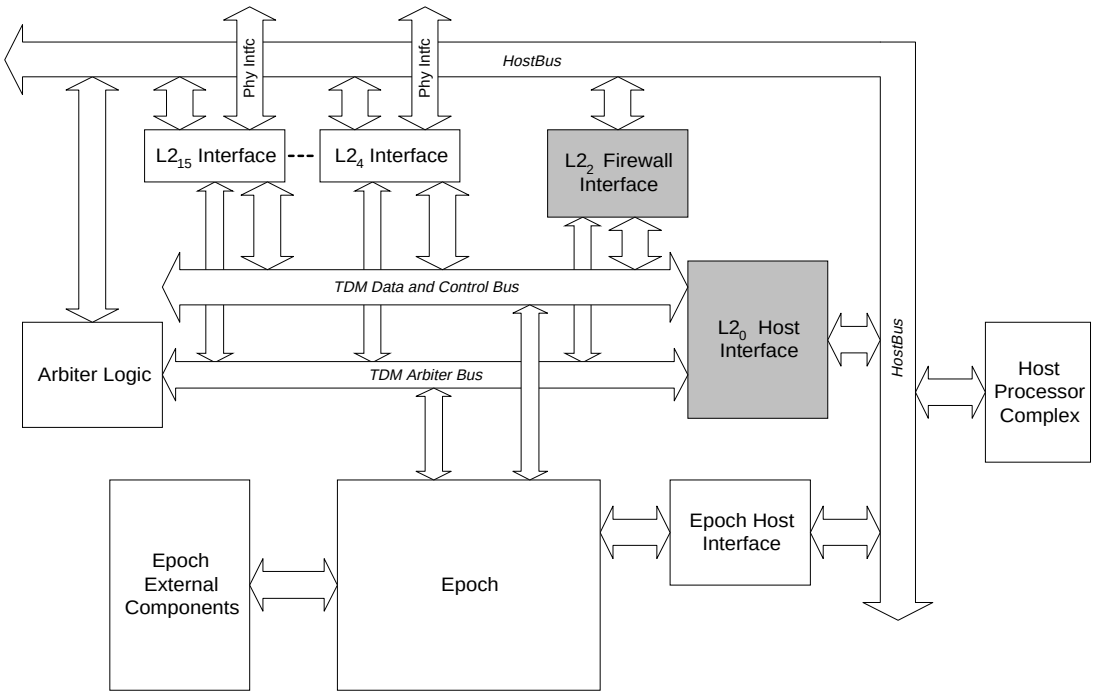


Figure 1-3: Level 1 Application Block Diagram - Host Processor Interface

In this application, the Host Bus controls and configures Epoch, Layer 2 interfaces and the Arbiter. The TDM Bus exchanges packets between Epoch and the other devices attached to the bus. In a typical situation, a packet arrives on a Layer 2 interface (L2port = n), which sends it to Epoch on the TDM Bus during the cycle allocated to L2port n by the Arbiter. Once switched by Epoch, the packet will come out on the TDM Bus during the cycle allocated to the destination L2port. Packets are exchanged over the TDM Bus in 64 byte chunks also referred to as buffers. Many packets such as TCP SYN/ACK packets are less than 64 bytes and can therefore be exchanged within a single TDM Bus timeslot.

When packets cannot be processed by Epoch, they can still be switched, but the sending device on the TDM Bus must provide the destination port number. For instance, an ARP message received at a Layer 2 interface is sent the host processor to update the ARP cache. This is accomplished by sending the entire Layer 2 frame to Epoch on the TDM Bus in bypass mode, indicating the processor port (L2port = 0) as the destination port. Sending the entire Layer 2 frame allows the host processor to determine the Layer 3 protocol used.

Lookup and Storage Elements

The external elements shown in Figure 1-2 include the RCP and four storage elements. The RCP performs the Layer 3 and Layer 4 lookup which then indexes into the associated data storage. The four storage elements are:

- **Packet Data Storage**–Contains the 64 byte packet buffers for all the packets waiting to be transmitted.
- **Packet Control Storage**–Contains the linked lists and control information for the buffers in the Packet Data Storage.
- **Packet Pointer Storage**–Contains the packet pointers for each of the eight queues assigned to each of the 16 TDM Bus interface ports.
- **Layer 3/Layer 4 Database Storage**–Contains the TDM Bus interface maps or portmaps, multicast control fields called MIAN, flow handles associated with specific microflows, default flow handles and DS flow handles. Typical access is through indices resulting from RCP Layer 3 and Layer 4 lookups.

Epoch Internal Architecture

The principal internal elements of the Epoch chip are shown in Figure 1-4. Note that this diagram is a simplified version of the one that appears in the *Epoch Data Sheet*. Host processor software interacts primarily with the Parse Engine (MLP), the Buffer Manager (BFM), the Packet Executive (PX), and the Register Set in the Statistics module (STM). Refer to the *Epoch Data Sheet* for detailed explanations for each block.

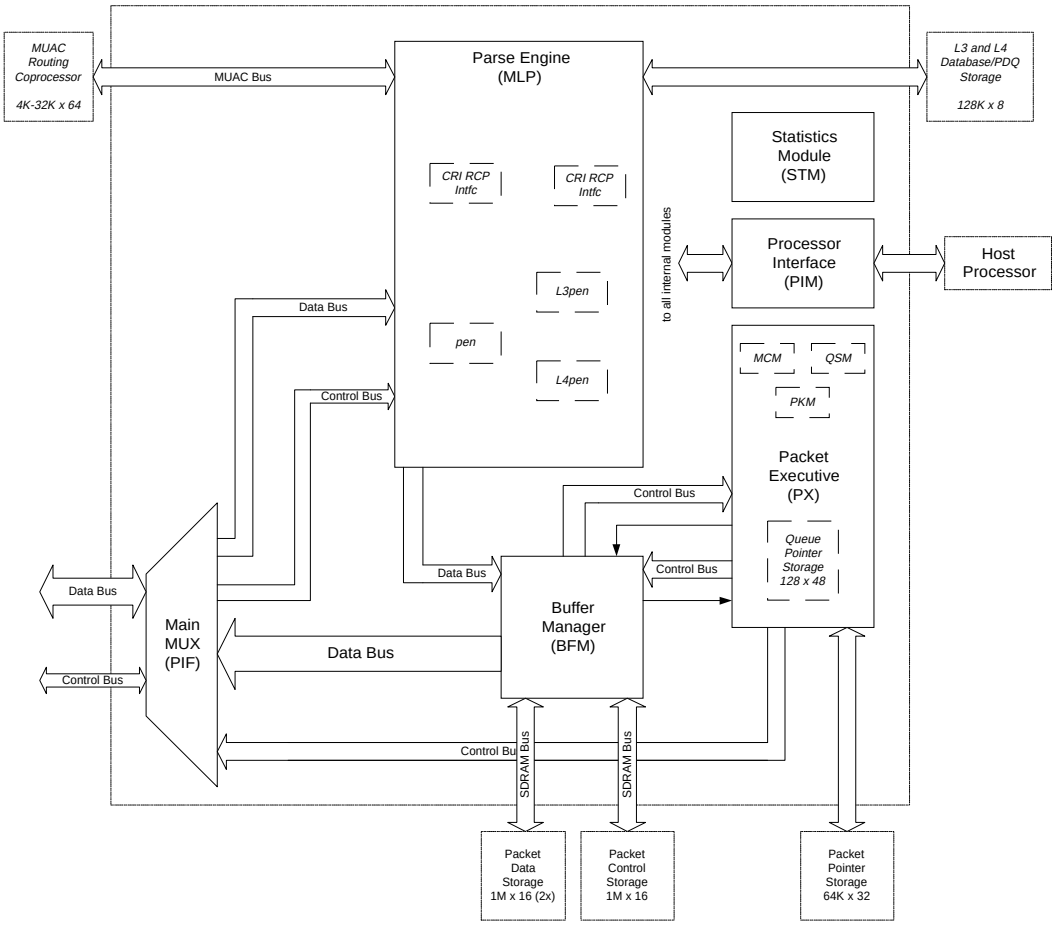


Figure 1-4: Epoch Block Diagram of Major Internal Elements

Parse Engine - MLP

The Parse Engine (MLP) consists of data sheet blocks PEN, L3PEN, L4PEN, CRI RCP Interface, and CRI SRAM Interface. The MLP has the following functions and characteristics:

- Handles special packets. For example, Layer 2 pass through packets, and re-injected packets.
- Parse packets. IPv4, TCP/UDP and IPX are parsed.
- Update fields such as TTL and Header Checksum
- Diverts packets with violations to the host TDM Bus interface. Example violations include expired TTL and MTU overflow.
- Determines the queue for buffering
- Determines the outgoing TDM Bus port bit map
- Determines the replacement DS field
- Perform packet divert and drop
- Assist in Layer 4 entry aging
- Coordinates RCP (Routing CoProcessor) access between the Epoch internal elements and the host processor port
- Provides a coordinated means to access data held in the Layer 3/Layer 4 associated data storage element

Buffer Manager - BFM

The Buffer Manager (BFM) functionality includes:

- Stores transmit buffers while TDM Bus interface ports are busy
- Stores buffers as they are received from the TDM Bus
- Performs multicast packet duplication
- Queues buffers to MLP directed queues
- Retrieves queued buffers for transmission as dictated by the PX
- Controls the external packet data and packet control storage

Packet Executive - PX

Packet Executive (PX) consists of data sheet blocks MCM, PKM, and QSM. The PX has the following functions and characteristics:

- Creates queue entries for unicast and multicast packets (MCM)
- Performs packet pointer manipulation (PKM)
- Determines service order of stored packets to be transmitted out of Epoch (QSM)

Statistics Module - STM

The Statistics Module (STM) block provides dynamic information about activity within the Epoch. There are Epoch internal counters that monitor such events as the total number of packets aborted or the number of aborts due to Layer 3 match failures.

SOFTWARE

A complete Layer 2/3/4 switch requires the use of several software modules as shown in Figure 1-5. The overall software package is responsible for processing various network messages to update Epoch's forwarding database and handling data streams that Epoch does not process. This software is responsible for other tasks including Layer 2 ARP and VLAN. The modules can be divided into several categories as follows:

- Basic TCP/IP protocols–IP, TCP, UDP, ICMP, IGMP
- One or more routing protocols–RIP, OSPF, DVMRP, BGP, PIM
- Quality of service–RSVP, Diff-Serv
- Management data–SNMP, MIB Manager
- Novell protocol–IPX
- Layer 2 processing–Ethernet, ARP/RARP, VLAN
- Optional initialization–DHCP
- Optional software–Epoch Monitor, Power-On-Reset (POR) testing, on-line diagnostics
- Optional software based routing–IPv6, DECnet, AppleTalk

Additional Elements

Although this document concentrates on software that is directly involved in the operation of the Epoch chip set, it is important to understand that a complete solution involves other elements. In a configuration as shown in Figure 1-3, the software resides in the block labeled Host Processor Complex. The choice of a routing protocol would depend on the system application. An edge router, as shown in Figure 1-1, could use BGP-4. The RSVP protocol could be included to communicate Quality of Service (QoS) information. This QoS information drives the Diff-Serv software block. The IPX software block handles IPX packets that the Epoch chip does not, for example, IPX packets with the checksum not equal to `0xffff`. The optional DHCP protocol can be used to determine the switch's IP address as well as other initialization and administrative functions. Finally, software can process other, low traffic, protocols such as DECnet, AppleTalk, and IPv6.

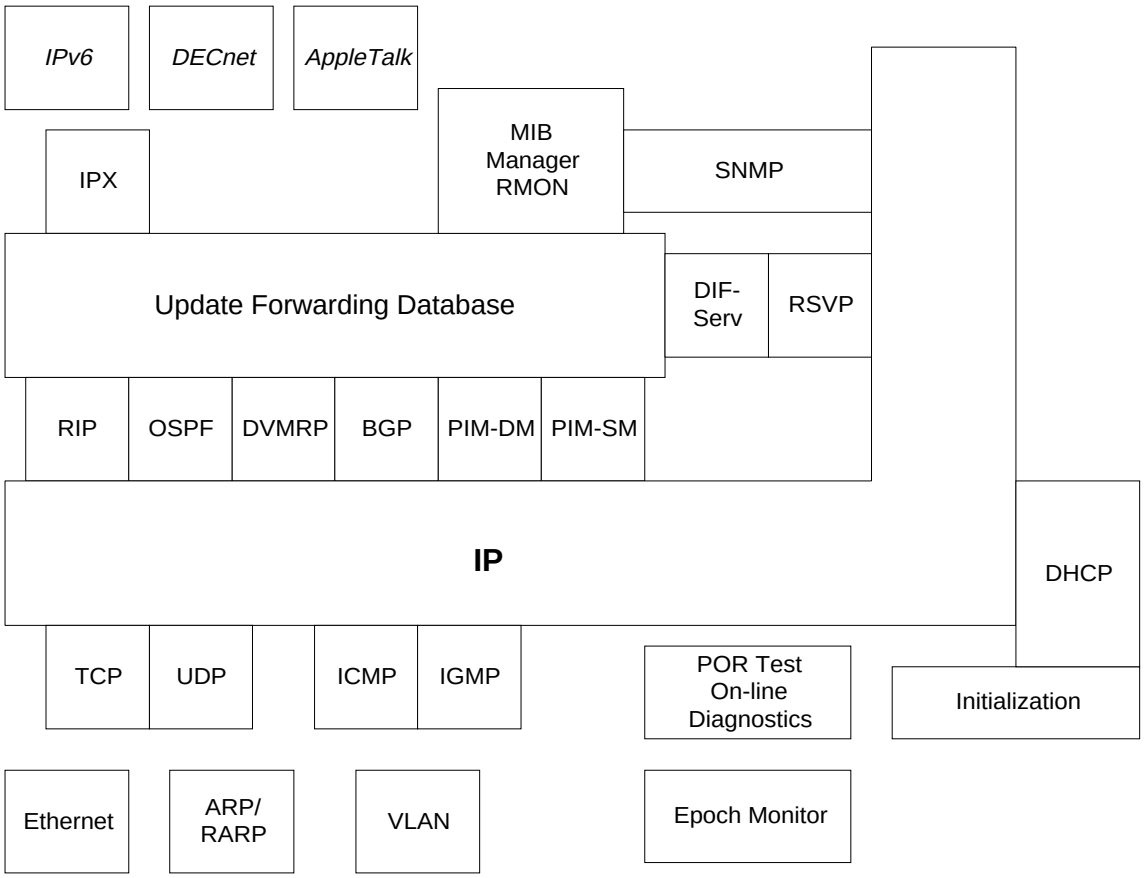


Figure 1-5: Major Software Blocks for Complete Layer 2/3/4 Switch

DOCUMENT CONTENTS AND TERMINOLOGY

Contents

Table 1-1 details the contents of each section in this manual.

Table 1-1: Document Contents

Section	Contents
Section 1: Overview	Provides an overview and defines one, of many, possible Epoch applications for clarifying discussions throughout the document. Gives an overview of the *Epoch Software Manual*. It also describes the notational conventions used in this manual and lists related MUSIC Semiconductors manuals and documentation of interest to programmers and hardware designers.
Section 2: Features	Presents detailed information on various Epoch features.
Section 3: Initialization and Configuration	Presents code fragments and gives a framework for the needs of power-on-reset and initial Epoch configuration settings. Furnishes implementation details for writing detailed C code.
Appendix A: Register Summary	Provides a quick reference to all Epoch registers.
Appendix B: Packet Header Layouts	Provides a quick reference to standard packet headers.
Appendix C: C Language Header Files	Provides the C programmer with header files to define register addresses by name, for instance.
Appendix D: External References	Lists references to standards applicable to Epoch operation.
Index	An detailed index listing all important features.

Terminology

Terms and Abbreviations

Table 1-2 lists non-obvious terms and abbreviations used in this manual.

Table 1-2: Terms and Abbreviations

Term	Context	Definition
Adjacency Pointer	Epoch Architecture	Value of the Layer 3 lookup index. Makes a link between the Layer 3 address lookup (IP destination address) and Layer 2 address table (for example Ethernet MAC, FR DLCI).
ARP	IETF	Address Resolution Protocol (RFC 826)
BAC	Diff-Serv	Behavior Aggregate Classifier or Classification
BFM block	Epoch Architecture	Buffer Manager. Major Epoch element (see Figure 1-4 and the *Epoch Data Sheet* section BFM).
Block Names	Epoch Architecture	See Figure 1-4 and MLP, BFM, PX, PIF, STM, PIM.
CIDR	IETF	Classless Inter-Domain Routing
`D bit`	Epoch Architecture	Flow Handle control bit. Determines if a packet is to be dropped or not (see Figure 2-2 in Features section).
Diff-Serv	IETF	Differentiated Services. See section Differentiated Services in Initialization and Configuration.
DS	IETF	See Differentiated Services. The DS field is the 8-bit field in the IP header formerly called TOS (see RFC2474/RFC2475, Figure B-1).
Flow Handle	MLP	16-bit value accessed through Layer 4 classification. Includes Replacement DS field, T8 bit (8/6 bit replacement size), R bit (request DS replacement), T bit (touch) and Queue handle (see Figure 2-2 in Features section).
`H bit`	MLP	Flow Handle control bit. Determines if a packet is to be diverted to the host port.

Table 1-2: Terms and Abbreviations (continued)

Term	Context	Definition
IPv4	IETF	Internet Protocol Version 4 as defined in RFC 791
IPX	Networking	Internet Packet Exchange. A Novell proprietary internet protocol.
LAN	Networking	Local Area Network
Layer 2	ISO/IETF	Data Link Layer
Layer 3	ISO/IETF	Network Layer (IP, IPX)
Layer 4	ISO/IETF	Transport Layer
L2port	Epoch Architecture	TDM Bus interface timeslot or port. The port number is determined by the Arbiter (see Figure 2-2 in Features section).
L3ad	MLP	Layer 3 Associated Data. Contains the IP TDM Bus portmap and MIAN for multicast.
L3CAM	MLP	Layer 3 32-trit ternary CAM
L3INDEX	Epoch Architecture	Layer 3 match index supplied with outgoing packet on TDM Control Bus
L4ad	MLP	Layer 4 Associated Data. See Flow Handle section.
L4CAM	MLP	Layer 4 64-bit binary CAM
L4INDEX	Epoch Architecture	Layer 4 match index supplied with outgoing packet on TDM Control Bus
Microflow	MLP	Group of IP packets with the same source and destination addresses, same TCP/UDP port numbers and same input physical port.
`MIAN bits`	MLP	Multicast Interface Authentication Enable. Bit in CREG0 register that controls dropping of multicast packets that arrive from other than the specified physical port.
MLP block	Epoch Architecture	MultiLayer Parser. Major Epoch element (see Figure 1-4 and the *Epoch Data Sheet*, sections PEN, L3PEN, L4PEN, CRI RCP Interface and CRI SRAM Interface).
PDQ	MLP	Port Default Queue. Flow handle table indexed by TCP/UDP port number when Layer 4 microflow lookup fails.
PIF block	Epoch Architecture	Port Interface multiplexer. This block manages the external TDM Bus (see Figure 1-4).
PIM block	Epoch Architecture	Processor Interface Module (see Figure 1-4 and the *Epoch Data Sheet*)
POR	Epoch Architecture	Power-On-Reset. The activation of the Epoch reset pin.
Portmap	MLP/Epoch Architecture	TDM Bus Interface map. Defines which Epoch output TDM Bus interface port(s) the packet is destined for. Each bit position in the 16-bit map corresponds to a TDM Bus interface port.
Punt field	MLP	TDM control bus field containing reason code for forwarding a packet to the Host.
PX block	Epoch Architecture	Packet Executive. Major Epoch element (see Figure 1-4 and the *Epoch Data Sheet* sections PKM, MCM, and QSM).
Queue handle	MLP	Five bit field. D bit (drop), H bit (host), and Queue number (see Figure 2-2 in Features section).
QoS	Networking	Quality of Service
`R bit`	Epoch Architecture	Flow Handle control bit. Determines if the DS field is to be replaced or not.
RCP	MUSIC Semiconductors	Routing CoProcessor - MUAC
Registers	MLP register	See Appendix A, Register Summary

Table 1-2: Terms and Abbreviations (continued)

Term	Context	Definition
STM block	Epoch Architecture	Statistics Module. Major Epoch element (see Figure 1-4 and the *Epoch Data Sheet* sections STM and other registers).
`T bit`	Epoch Architecture	Flow Handle control bit. Determines if a packet has been processed by Epoch (see Figure 2-2 in Features section).
`T8 bit`	Epoch Architecture	Flow Handle control bit. Determines whether 8 or 6 DS bits are replaced if the R bit is set (see Figure 2-2 in Features section).
TDM Bus	Epoch	Time Division Multiplexed bus. Epoch's interface to all Layer 2 devices. Port zero is the Host TDM Bus interface, not to be confused with the Host Processor Interface port used to access Epoch internal registers.
VoIP	Networking	Voice over IP
WAN	Networking	Wide Area Network

Notations

Register and Field Notation

Single or multiple bit fields are specified by using by a pair of numbers in square brackets ([]) separated by a colon (:) and are inclusive. For example, RegisterName[6:4] and RegisterName[2] specify bit 4 through 6 and bit 2 of RegisterName respectively.

Numbering

All numbers are decimal or hexadecimal unless otherwise indicated. The prefix 0x and special font indicate a hexadecimal number. For example, 19 is decimal, but `0x19` and `0x19A` are hexadecimal. Otherwise, the base is indicated by a subscript; for example, 100_2 is a binary number.

Reserved and Not Implemented Bits and Software Compatibility

Certain register addresses as well as code values are marked as **Reserved**. To maintain compatibility with future Epoch products, these address locations and codes should not be used.

In addition, certain register bits are marked as **Not Implemented**. To ensure future compatibility, these bits must be treated as having a future function. The behavior of these bits should be considered undefined and unpredictable. The following guidelines should be used:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

Any software dependence upon the state of reserved bits or addresses must be avoided. Programs that depend upon reserved bits or values risk incompatibility with future Epoch products.

RELATED MATERIAL

The documents listed below contain additional material related to Epoch. Documents may be obtained online from the MUSIC website, http://www.music-ic.com.

- *Epoch Data Sheet*
- *MUAC RCP™ Data Sheet*
- *AN-N25 Fast IPV4 and IPV4 CIDR Address Translation and Filtering Using the MUAC Routing CoProcessor (RCP)*
- *AN-N27 Using MUSIC Devices and RCPs for IP Flow Recognition*
- *Epoch Reference Design*
- *Epoch API Software Manual*

Features

INTRODUCTION

This section contains discussions of the use of the key features and functions for typical Epoch applications. Follow-on sections outline, in a more precise manner, with code fragments, how to use the Epoch with a level of detail sufficient for writing actual software modules. A more complete package of evaluation hardware and software is available. For more detail, see Related Material on page 1-11, *Epoch Reference Design*.

KEY FEATURES

Microflows

A key feature embodied in Epoch is the Microflow feature. Its definition is similar to a combination of the SESSION and FILTER_SPEC RSVP objects. A Microflow associates IP packets and applies a specific treatment to these packets. Microflows are uniquely identified in Epoch by their IP source and destination addresses, TCP/UDP source and destination ports, and L2port. The Microflow identifier is referred to as Microflow ID in the rest of the document.

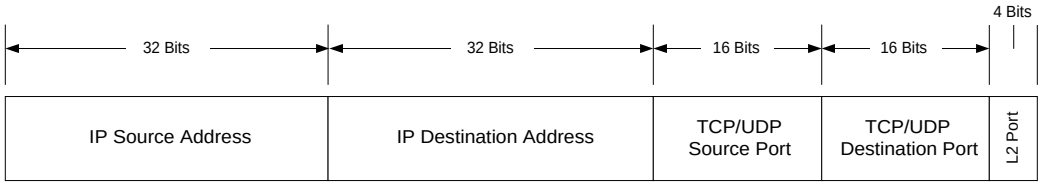


Figure 2-1: Microflow Identifier

Flow Handles

A Flow Handle is a 16-bit word containing processing options that are key to delivering multiple levels of service to groups of packets. The Flow Handle layout is shown in Figure 2-2. It includes the following fields:

- **Replacement DS Field**–Replaces the packet DS field depending on value of `T8 bit` and `R bit`.
- **`T8 bit`**–If set, all 8 bits of the DS field are replaced, otherwise only the upper 6 bits are replaced. In both cases, the `R bit` must be set.
- **`R bit`**–If set, the packet DS Field is replaced by all or part of the DS field in the Flow Handle.
- **`T bit`**–The touch bit is only used in Flow Handles stored in the Layer 3/Layer 4 Database and is set to 1 every time the Flow Handle is accessed by the RCP. It is used for entry aging.
- **Queue Handle**–Determines the treatment applied to the packet. It consists of the following sub-fields:
  - **`D bit`**–If set, the packet is dropped.
  - **`H bit`**–If set, the packet is diverted to the Host Processor Port, L2port 0.
  - **Queue**–The number of the queue where the packet is to be queued. Zero is highest priority, seven lowest.

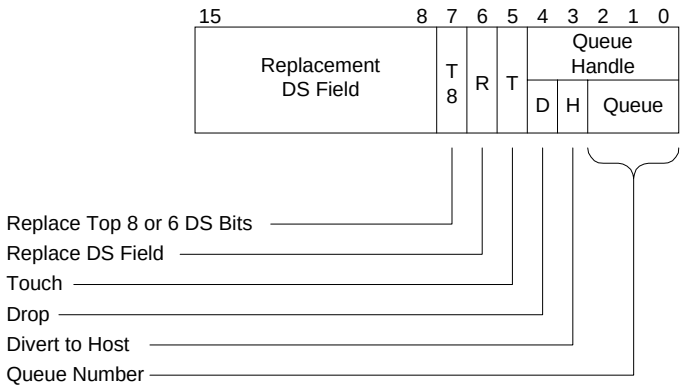


Figure 2-2: Flow Handle

Flow handles are accessed in the Layer 3/Layer 4 Database using a Microflow ID, DS field or TCP/UDP port number depending on the current operating mode and packet header. Epoch exception handling registers use a subset of the Flow Handle format. For instance, the `T bit` is not relevant and is not used.

Classification

Epoch can classify incoming packets in one of the following two ways:

- Microflow Classification
- Behavior Aggregate Classification (BAC)

Each incoming port can be configured independently to use either classification. Classification applies only to IPv4 packets without options and not re-injected.

Fragmented packets and non TCP/UDP IPv4 packets are treated separately and each form a class of their own, and their processing is governed by the registers DefaultFragPackFlowHandle and DefaultUnknownL4ProtoFlowHandle respectively.

Microflow Classification

If a port is configured in **Microflow** mode, the IPv4 Layer 3 and Layer 4 headers are parsed by Epoch and assembled into a Microflow ID as shown in Figure 2-1. The Microflow ID is used to lookup the Layer 3/Layer 4 Database and retrieve the Flow Handle associated with the Microflow.

If the Microflow ID is not found, the **Default Queue** method is used as follows:

- The two 16-bit TCP/UDP source and destination port numbers are used as indices into the 64K Port Default Queue (PDQ) table to retrieve two Flow Handles. The Flow Handle with the highest priority, the lowest queue number, is used.
- Both Default Queue and Microflow searches are performed concurrently, the latter having precedence.

The flowchart for this processing mode is shown in Figure 2-3.

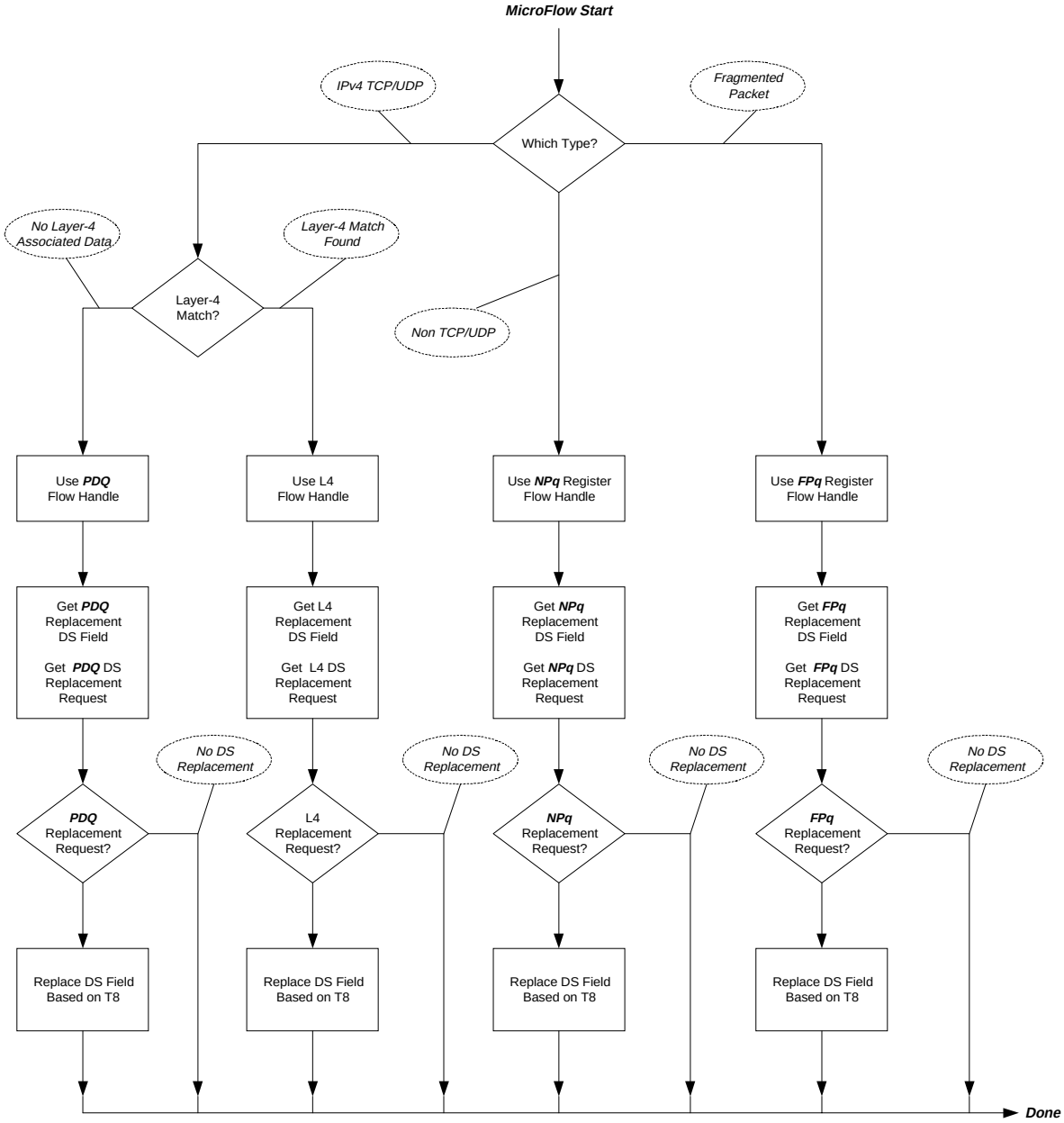


Figure 2-3: Differentiated Services Processing - Microflow Mode

Behavior Aggregate Classification - BAC

If a port is configured for BAC mode, the 8-bit DS field is extracted from IPv4 packets arriving at this port. The field is used as an index in the 256 element BAC Table, contained in the Layer 3/Layer 4 Database, to retrieve a Flow Handle. The Flow Handle, shown in Figure 2-2 includes a DS field, which can be used to replace the existing value in the packet. Figure 2-4 shows the flowchart of this processing mode.

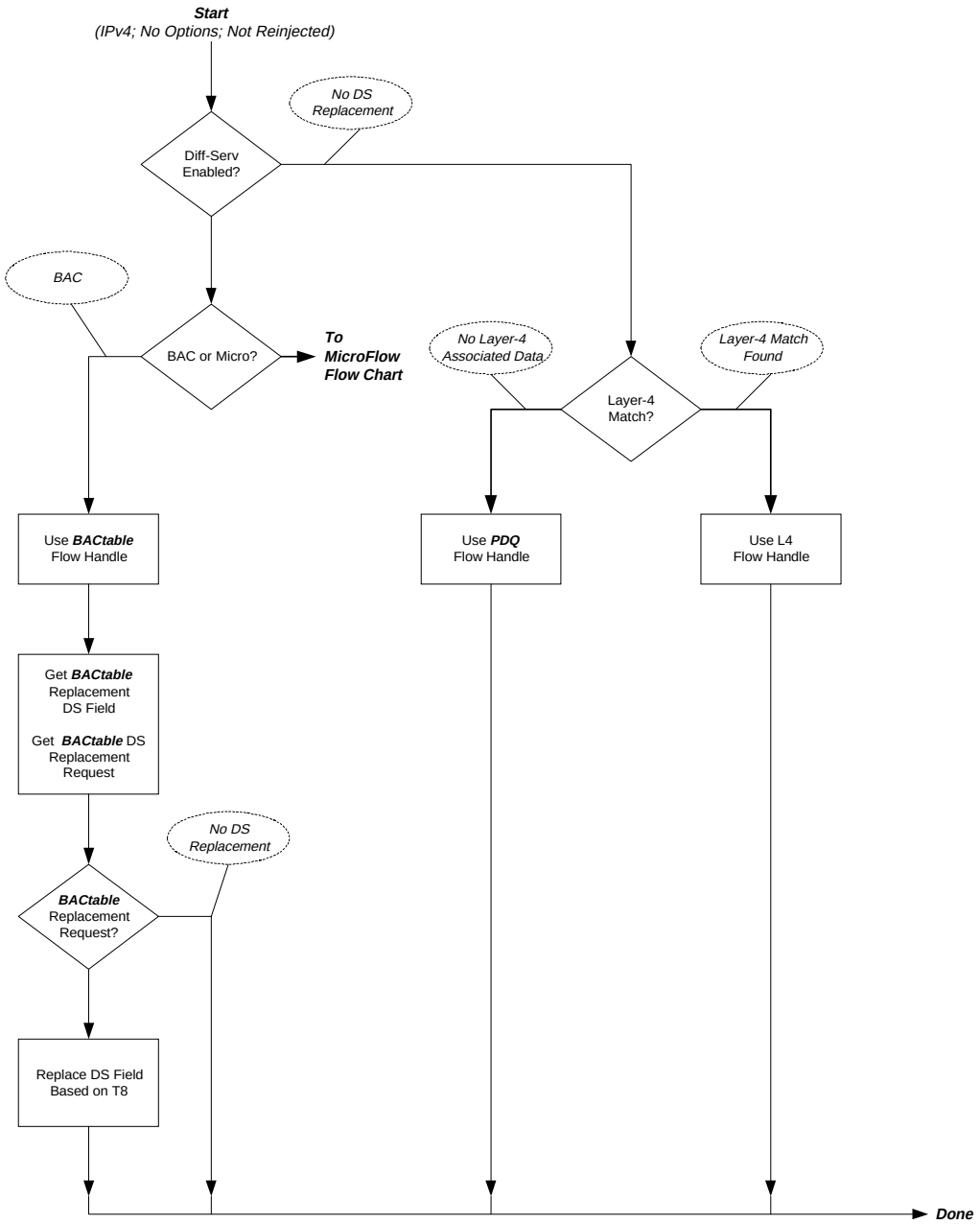


Figure 2-4: Differentiated Services Processing - BAC Mode

PACKET DESTINATIONS

Epoch can be configured to forward selected packets to the host TDM Bus interface based upon specific Layer 3 destination addresses. The Layer 2 interfaces also can forward selected packets to the host TDM Bus interface based on specific Layer 2 addresses. In any Epoch-based product, there are packets for which the destination address is the switch itself. These messages include router protocol messages, for instance BGP or OSPF, ICMP messages, and Layer 2 messages. These messages must be processed by the host processor.

Router protocol messages such as RIP router advertisement are interpreted, and possibly used to update the forwarding database within the Epoch chip set. There are many router protocols used in different applications. Figure 1-5 shows some of the more important variants including RIP, OSPF, DVMRP, BGP, and PIM.

For Epoch-based products that support Ethernet at Layer 2, messages important to switch operation include ARP and RARP packets. The information from these packets can be used to update a Layer 2 ARP cache. This cache is external to Epoch, but it can use the Layer 3 destination address lookup index provided by Epoch to retrieve the Layer 2 destination address from the ARP cache. Any further discussion of ARP cache details is beyond the scope of this document.

QUEUING ALGORITHMS

The Epoch chip offers, for each TDM Bus interface port, a choice of two selectable queuing algorithms: Weighted Round Robin or Priority Order. Epoch handles up to eight queues per TDM Bus interface port. The ability to schedule queue servicing at different priorities is a key feature needed to implement Quality of Service (QoS) capabilities in a network switch.

- **Weighted Round Robin**–Queue zero, the highest priority queue, receives half of the cycles, queue one receives a quarter of the cycles, and so forth. The scheduler is work conserving, meaning simply that no packet slot on a port is left idle as long as there are packets in some queue for that port.
- **Priority Order**–Queues with higher priority are serviced before lower priority queues as long as there are packets to be serviced: Queue zero is emptied before Queue one is serviced, then Queue 1 is emptied before Queue two is serviced, and so forth. If another packet arrives in a higher priority queue it takes precedence and is transmitted first.

DIFFERENTIATED SERVICES

The Epoch Differentiated Services (Diff-Serv) features are based upon IETF RFC2474 and RFC2475. The following excerpt from RFC2475 provides a broad Diff-Serv model description:

The differentiated services architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single DS codepoint. Within the core of the network, packets are forwarded according to the per-hop behavior associated with the DS codepoint.

As can be seen, a network device situated at the boundary of a DS network, or domain, needs to classify entering packets and possibly condition the traffic, which can involve metering, marking, policing and shaping. The following definitions, taken from RFC2475, will help understand their relationships:

- **Classifier**–Entity which selects packets based on the content of packet headers according to defined rules.
- **Traffic conditioning**–Control functions performed to enforce rules specified in a Traffic Conditioning Agreement, including metering, marking, shaping, and policing.
- **Metering**–The process of measuring the temporal properties (e.g., rate) of a traffic stream selected by a classifier. The instantaneous state of this process may be used to affect the operation of a marker, shaper, or dropper, and/or may be used for accounting and measurement purposes.
- **Marking**–The process of setting the DS codepoint in a packet based on defined rules; premarking, re-marking.
- **Shaping**–The process of delaying packets within a traffic stream to cause it to conform to some defined traffic profile.
- **Policing**–The process of discarding packets (by a dropper) within a traffic stream in accordance with the state of a corresponding meter enforcing a traffic profile.

Figure 2-5 is inspired by Figure 1 in RFC2475 and is a graphic depiction of classification and conditioning.

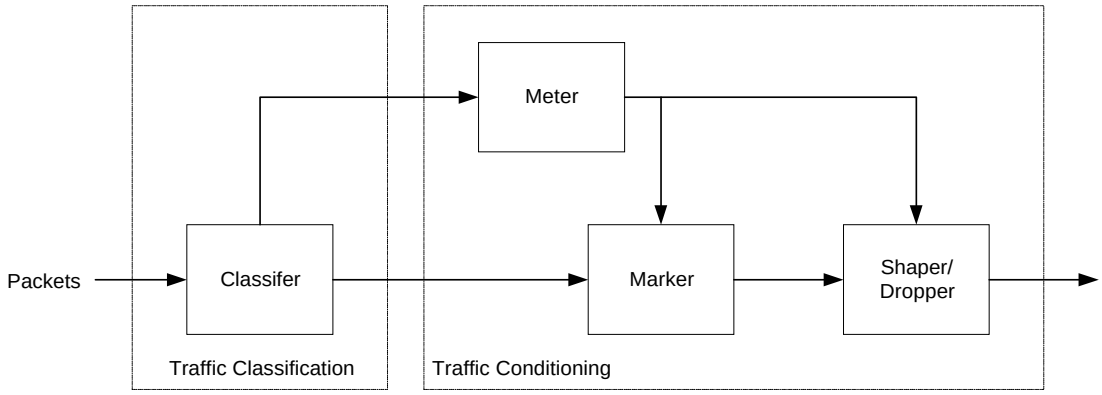


Figure 2-5: Differentiated Services Packet Processing

Key Epoch features are targeted at Differentiated Services and perform or facilitate the functions outlined above. The following sections explain how to implement DS functions using these features.

Classification

At the edge of a non-DS and a DS network, a non-DS marked packet can be classified by Microflow using the header fields shown in Figure 2-1. If packets are already DS marked, BAC can be used, in which case packets are classified by their DS field value. See section Classification on page 2-2 for a more complete description of Epoch classification features.

Traffic Conditioning

Metering

Metering can be implemented by a user supplied device attached to a TDM Bus port. By using Epoch intercept or snooping features, packets can be respectively diverted or duplicated to the metering port. L3INDEX and L4INDEX are available to the intercept/snoop port as well, allowing the metering device to retrieve and update data associated with a packet destination address or Microflow ID.

The metering device can take action about packets that exceed their class traffic profile and delay or drop them. Or it can provide the host processor with information about these packets, which can then modify the Layer 3/Layer 4 Database to change the flow handle for that packet class.

Marking

Classification yields a flow handle, which includes a Replacement DS field and option bits, `R bit` and `T8 bit`. This allows packets in a given class to be marked with a given DS field. When BAC mode is in use, this amounts to DS re-marking. The flow handle can be changed by the host processor in response to administrative commands or to metering feedback.

Shaping

Epoch can prioritize packets but can not delay them, as is the case with shaping. If shaping is required, then packets must be diverted to a shaping device attached to a TDM port. The shaping device is under the control of a metering device as described above. Once released from the shaper, packets can be re-injected in Epoch for switching.

Without a shaping device, relative packet class traffic shaping can be achieved by changing its flow handle priority.

Policing

Packets can be readily dropped by using the `D bit`. If the host processor, with the help of the metering device, detects that a class of packets exceeds it traffic profile, it can set the `D bit` in that packet class flow handle.

LAYER 4 PROCESSING

One of the most significant features of Epoch is its Layer 4 processing capabilities and options. Packets can be classified then routed and prioritized according to this classification. This processing has two main applications: fine-grained Quality of Service, and security in firewall applications, both involving Epoch and its Host Processor to various degrees. At one end of the spectrum, all packets are directed towards the Host Processor, while at the other end, all are automatically processed by Epoch.

Host Processor Only

In host processor only Layer 4 operation, the processor examines every single packet entering the switch. This is not a recommended mode of operation since it essentially defaults to a software-based router whose bandwidth is governed by the processor speed. However, in a system that is acting as a firewall, it is a common practice to start by denying or examining every packet. In this case, the Port Default Queue (PDQ) table in Epoch has all host intercept bits (`H bit`) set, forcing the host processor to examine every packet that traverses the firewall router.

Epoch Only

In this mode of operation, all flows are allowed to pass through, with packet processing performed by the processor only for special case packets, also called punted packets. Punted packets are packets that Epoch can not process without processor intervention. IP packets with options, IPv6, and packets that need fragmentation fall into this category. The bandwidth utilized in the Epoch approaches the theoretical maximum while the CPU requirements are minimized. No host bits are set in the default queue table and no Layer 4 CAM entries are used.

LAYER 4 LEARNING AND AGING METHODS

Layer 4 learning is the process by which a new Microflow ID and its associated data are entered into the Layer 3/Layer 4 Database. Its counterpart process, Layer 4 aging, involves the removal of inactive Microflows from that database. This process is necessary to keep available space in the database for new Microflows.

Layer 4 Learning

Once a Microflow has been learned by entering its Microflow ID and associated data into the Layer 3/Layer 4 Database, Epoch can automatically process packets in this microflow at wire speed without host processor intervention. Deciding which Microflows should be learned is therefore an important decision, which affects the performance level achieved with Epoch.

One method of learning Microflows is to set the H bit in PDQ Flow Handles corresponding to selected well known TCP/UDP ports, while setting Flow Handles corresponding to ephemeral port with the lowest priority to ensure the well known port Flow Handle is retained. With this method, whenever an IPv4 TCP/UDP packet with one of the selected well known ports enters Epoch, if it doesn't belong to an existing Microflow, then it is diverted to the Host Processor, which can then create a new Microflow entry.

Certain well known TCP/UDP ports requiring no specific or differentiated processing can have their corresponding Flow Handle in the PDQ table set with the desired processing options and the `H bit` reset. All packets with that port number are then processed automatically by Epoch without any Host Processor intervention.

Another consideration is the fact that longer flows require less relative learning overhead since only the first packet of a flow needs to be examined.

Deciding which Microflows should be learned is therefore the result of a compromise between service level requirements, performance, security, and resources.

Depending upon the degree of security desired in an edge router utilizing the Epoch chipset, it is recommended that processor–directed Layer 4 learning is executed only on flows that are deemed to be long-lived or a potential security risk. The MCI paper "Wide Area Internet Traffic Patterns and Characteristics" by Thompson, Miller, and Wilder examines traffic characteristics in the Internet and determines that the average TCP flow contains 15-20 packets. Selecting a processor capable of handling 1 of 15 to 20 packets entering the switch would enable excellent security and flow optimization. However, according to the MCI paper, even just optimizing the 5% longest flows yields performance

benefits for over 30% of the total traffic traversing the router, which would greatly decrease processor requirements. Long flows that would justify Layer 4 CAM entries for flow optimization would be flows from multimedia applications including RealAudio, video, and packet voice. Security sensitive flows that would justify processor examination would be applications flows such as POP (Post-Office Protocol), SendMail, and Telnet.

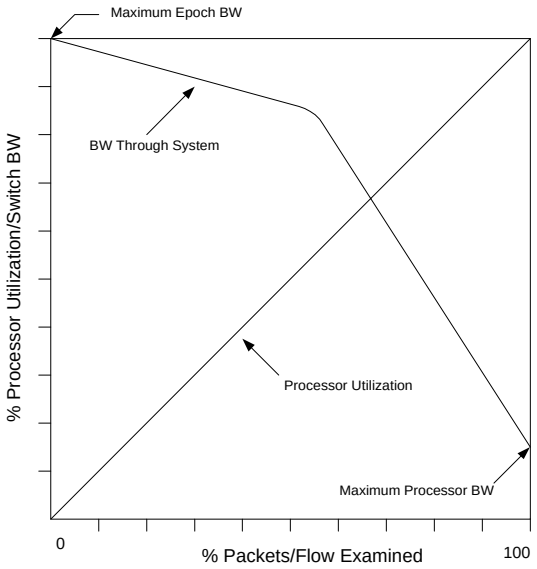


Figure 2-6: Hypothetical Bandwidth Utilization

A rough estimate of the processor MIPS required in a WAN edge router utilizing the Epoch and acting as a firewall follows. The example router has an ATM OC-3 link, yielding 155 Mbps in each direction. Using a worst case scenario of 100% 40 byte packets (not unreasonable, since about 40% of Internet traffic is 40 byte TCP SYN and ACKs) and neglecting ATM cell overhead, about (155 Mbps*2)/(40 Bpp*8b/B), or about 969,000 packets per second can traverse the link. If the processor needs to examine on average 1/15 of the packets, it must examine about 64 kpps. Using a rough estimate of 200 instructions per packet, we arrive at a MIPS requirement of about 12.5 MIPS for one OC-3 link. For a switch configuration with eight OC-3 links, we need a processor with 100 MIPS. Therefore, a low to mid-range PowerPC, Pentium or MIPS could handle this configuration. Router applications that are not as security or quality-of-service sensitive will require less MIPS. Hands–Off Layer 4 Classification could probably get away with a processor with 10-20 MIPS of performance, if not less. Some additional MIPS will have to be allocated if the processor is to calculate new routing tables and perform aging; however, the bulk of the MIPS are required for examining the packets.

Layer 4 Entries Aging

Microflows typically exist for a finite amount of time, some statistics indicating a 10 to 20 seconds life. Microflow aging must then be performed so that inactive entries can be removed from the Layer 3/Layer 4 Database to free up space for new entries. Epoch helps this process by setting the Flow Handle Touch bit whenever a packet matches the Microflow ID of that Flow Handle.

The host processor can use this Epoch feature by checking at regular intervals which Flow Handles have been touched, as indicated by a Touch bit set to 1. If a Flow handle hasn't been touched, then the Microflow can be removed immediately or an associated time-out counter incremented and the Microflow removed once the counter reaches a chosen threshold. The processor must then reset all Touch bits so that Flow Handles touched during the next time interval can be identified.

It takes approximately 50 clock cycles to read or write a Flow Handle and another 50 clock cycles to delete the CAM entry. Therefore, the processor needs at least 100 clock cycles to age a CAM entry. It can be more, depending on the processor and processor bus architecture. Based upon this, aging 1000 flows takes approximately 1.5 millisecond.

MULTICAST FEATURES

Epoch is compatible with all IP Multicast protocols and can be configured on a per-port basis to handle different forwarding protocols. The basic structure of a multicast address lookup by Epoch is: [Source Address, Destination/Group Address]. This is a flat lookup that does not allow Source Address subnetting. This scheme is ideal given the current level of multicast usage. There are several features that help facilitate efficient multicast forwarding.

Multicast TDM Bus Interface Authentication Number - MIAN

Most multicast routing protocols require source port authentication in addition to matches on source address and group address matches. This is necessary because certain network topologies allow routers to be multi-homed. Figure 2-7 shows an example network topology in which source port authentication is needed. The multicast group "G1" has members in LAN's 0-3 all listening to the G1 source "S". When S sends a packet to G1, router R0 receives the packet first on port 0 and multicasts it out ports 1 and 2. The problem arises when router R1 receives multicast packets of the form [S,G1] which it must forward out port 0 to LAN 3. Because LAN 1 and LAN 2 both receive identical [S,G1] packets from router R0, router R1 will receive *2 copies of the same packet*: one on port 1 and one on port 2. If router R1 does not perform interface authentication it will send the same packet out twice to LAN 3.

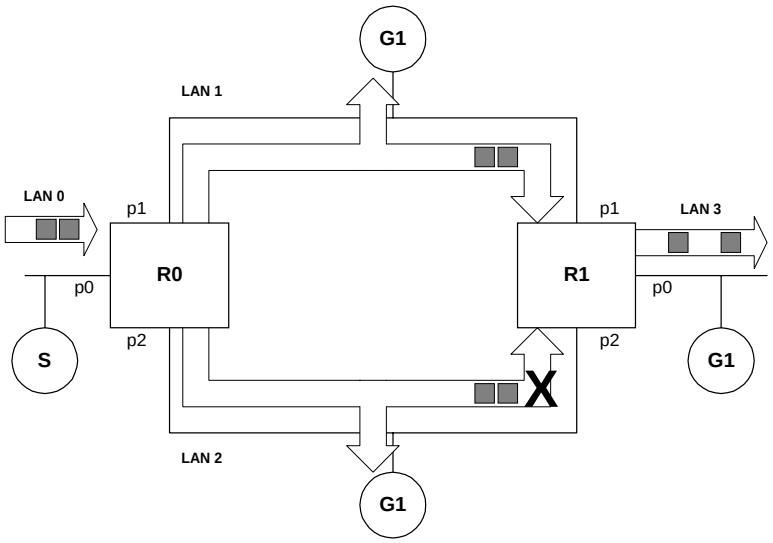


Figure 2-7: Potential Undesired Packet Duplication

This is an undesirable condition because it wastes bandwidth and also can cause problems with many applications. By choosing one interface to receive [S,G1] packets from, only one copy of the packet is propagated to LAN 3. In the example above, port 1 is the authenticated port.

Epoch has a feature called Multicast TDM Bus Interface Authentication Number (MIAN), which allows multicast source port filtering. Multicast groups are entered in the RCP by the combination [Source Address, Group Address]. The associated data is made of a port map and MIAN entry. Both are stored at the same relative locations, the RCP index, in their respective storage. The portmap is the same used for unicast entries, while the MIAN 4-bit entry contains the TDM Bus port number allowed to receive packets for that Multicast. Figure 3-16 shows the format of a Multicast entry in the RCP, and Figure 3-17 shows the location of the associated data. This source port multicast filtering feature can be turned on or off for each source port individually through the CREG0 register.

Multicast TTL Thresholding

Most multicast algorithms, such as DVMRP, can cause packet flooding situations in which multicast packets are sent to all interfaces. In the case of a WAN edge router, it is desirable to be able to block internal multicast floods from "leaking" outside the LAN. Epoch includes such a feature: Each TDM Bus port has its own TTL threshold, which prevents any multicast packet with a TTL lower than that threshold from being transmitted over that port. Offending packets are discarded and CntTTLThreshAbortCounter incremented. Different ports can have different thresholds.

TTLOFF PROCEDURES

Normal IPv4 packet forwarding involves decrementing the packet TTL upon receipt of the packet. For example, an IPv4 packet entering the Epoch with TTL of 20 would exit with TTL of 19. The Epoch parser offers the capability of enabling or disabling the decrementing feature on a per input port basis. For example, if TTLoff is enabled on port 5, a packet arriving on port 5 with a TTL of 20 would leave the Epoch with a TTL of 20 while packets arriving on all other ports would have their TTL decremented. There are several applications of this feature. One application is a system containing Epochs in a cascade or ring configuration. By enabling TTLoff on each Epoch input port connected to the output port of another Epoch, a packet can traverse multiple Epoch's while having its TTL decremented only once, in the first Epoch chip.

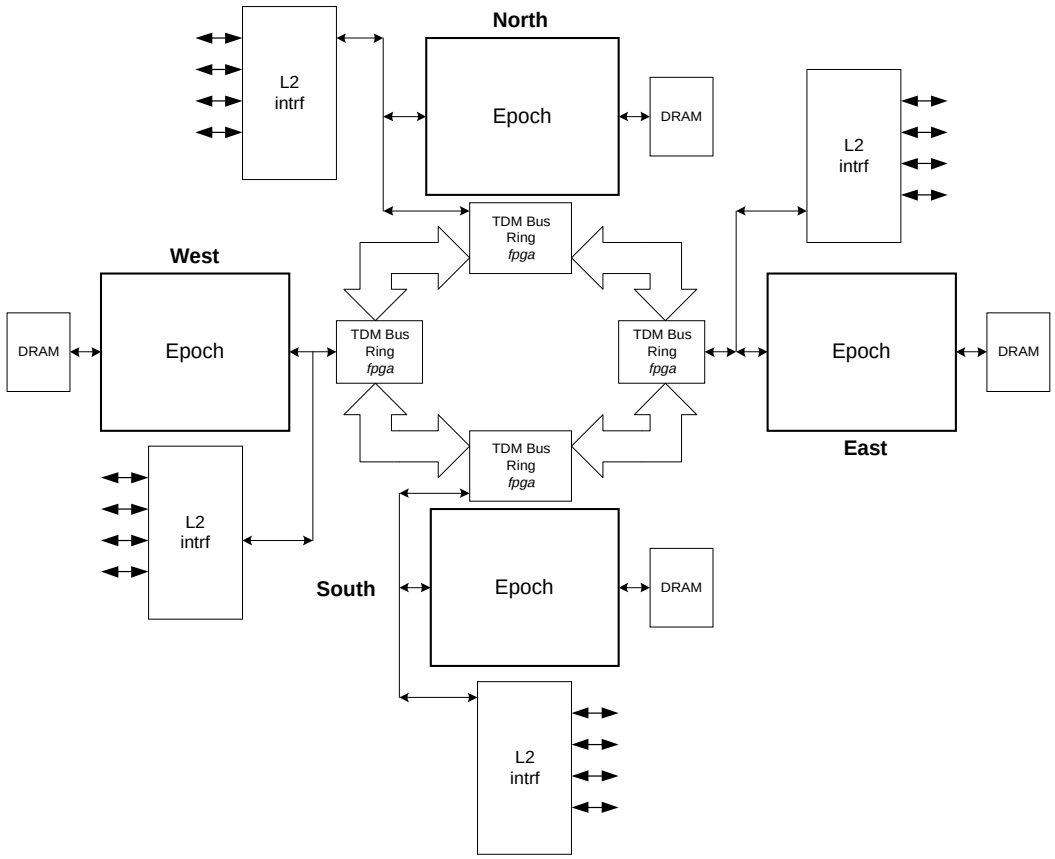


Figure 2-8: Ring Configuration

Figure 2-8 shows four Epoch chips connected in a ring configuration. This configuration could be blocking or non-blocking, depending upon how much bandwidth is allocated on the inter-Epoch ring links. The routing table configuration could be performed for each Epoch as if they were independent routers. Doing this would enable optimal forwarding that would be adaptive to traffic loading on the ring configuration. Each Epoch has TTLoff enabled on its ports connected to other Epoch chips while TTLoff is disabled on ports connected to Layer 2 interfaces.

As an example, an IP packet with TTL of 11 arriving at the Layer 2 interface of Epoch South, and ultimately destined for a network connected to Epoch North, is parsed and its TTL is decremented to 10 by Epoch South. Using its routing table, Epoch South transmits the packet to Epoch East, which doesn't decrement the TTL field since TTLoff is enabled on the port connecting it to Epoch South. Similarly, the packet is forwarded to Epoch North without modification of its TTL field. Epoch North then forwards the packet to its appropriate Layer 2 interface port. Altogether, TTL is decremented only once, upon entering the Epoch ring.

In a standalone system using Epoch, the TTLoff feature typically is not used. There may be novel applications in which this feature can be exploited as above including server load balancing.

NETWORK MANAGEMENT

Statistics Gathering

There are a number of events that cause Epoch to abort packet processing. Epoch maintains Event Counter registers for each of these events. Each Event Counter is associated with an Event Count Threshold register, which generates a maskable interrupt when the threshold is crossed. The register names and their functions are listed in Table 2-1.

Table 2-1: Epoch Event Counters

Event Counter	Event Count Threshold	Description
CntOutOfBuffers	CntOutOfBuffersThresh	Number of buffer full aborts
CntOutOfPacketPointers	CntOutOfPacketPointersThresh	Number of packet pointer exhausted aborts
CntFifoOverflow	CntFifoOverflowThresh	Number of multicast FIFO overflows
CntPacketTooBig	CntPacketTooBigThresh	Number of oversize (>64K-1) packet aborts
CntPacketTooSmall	CntPacketTooSmallThresh	Number of undersize (<40 IP and IP re-inject, <30 IPX) packet aborts
CntL2Abort	CntL2AbortThresh	Number of packets aborted by L2ABORT pin being asserted
CntTTLThreshold	CntTTLThresholdThresh	Number of multicast packets aborted because TTL threshold exceeded on ALL destination ports. Note multicast packet only counted once, not once for each destination port.
CntL3FilterAbort	CntL3FilterAbortThresh	Number of packets dropped because port bitmap set to 0 in associated data RAM. Note packet is still counted as dropped even if it is snooped or diverted to host.
CntL4FilterAbort	CntL4FilterAbortThresh	Number of packets dropped because Drop bit set in L4 flow handle. Note packet is still counted as dropped if it is snooped or diverted to host.
CntTotalAborts	CntTotalAbortsThresh	Total number of packets aborted

Interrupts

The statistics threshold registers listed above can be used to independently trigger a processor interrupt. Each threshold can be set to any value desired through the processor port. When a statistics counter register passes the value of the associated threshold register, an interrupt is triggered. For example, if the threshold is set to `0x000f` and the counter goes from `0x000f` to `0x0010`, the Epoch interrupt is activated. Each interrupt is individually maskable through the Interrupt Mask register.

Reading Statistics Registers and Wrap Behavior

The statistics counters are sticky; when a statistics counter is read through the processor port, it is reset to zero. When a statistics counter is at `0xffff` and is then incremented, it wraps to `0x0000` instead of saturating at `0xffff`.

Management Information Bases

Epoch includes features which help in implementing Management Information Bases (MIB) objects.

RMON

RMON I deals essentially with Layer 2 statistics and must be implemented by circuitry external to Epoch. This circuitry maintains counters for Layer 2 interfaces such as Ethernet interfaces.

RMON II specifies monitoring for Layer 3 and above. Although Epoch classifies packets at the Layer 3 and Layer 4 levels, it does not provide enough information to maintain all RMON II counters. At Layer 3, Epoch matches the packet destination address, returning L3INDEX on its Control Bus, but it does not match the source address, which is needed to update the corresponding RMON II rmon.nlHost.nlHostTable.nlHostEntry. Similarly, Epoch Layer 4 classification is too specific to be used in maintaining source/destination address pairs in rmon.nlMatrix.nlMatrixSDTable and rmon.nlMatrix.nlMatrixDSTable.

To provide full RMON II capabilities, specialized circuitry or the host processor need to be used in conjunction with Epoch source and/or destination port snooping.

Vendor Specific MIBs

Vendors can register with the Internet Assigned Numbers Authority (IANA) to obtain a private enterprise code in the iso.org.dod.internet.private.enterprise branch of the Structure of Management Information (SMI) tree. A registered vendor can then define product specific MIBs. These MIBs could include the Epoch counters described in section Statistics Gathering on page 2-11.

IPX FEATURES

The key Epoch features involved in IPX packet processing at wire speed are:

- Drop all incoming packets with a transport control field greater than fifteen.
- Divert to the host all packets with a checksum that is not `0xffff`.
- Divert to the host all packets with a packet type equal to 20 (`0x14`).
- Replace all source and/or destination node fields that are zero with the value in register IPXattach-PortN, where N is the incoming TDM Bus interface port number.
- Increment the transport control field.
- Perform a Layer 3 lookup for the destination TDM Bus interface port number.
- Use the DefaultIPXFlowHandle register for queuing information.
- Perform other processing such as Host Divert or Source/Destination Snooping as requested.

Figure B-5 can be used as a reference for the IPX packet layout.

IPX Initialization

Epoch processes IPX at the Layer 3 level only. Epoch does not examine socket numbers and microflows in IPX as it does with IPv4. This is to keep IPX processing simple because IPX is a LAN protocol that is not growing in usage. Since it is a LAN protocol where bandwidth is available, complex queuing is most likely unneeded. Sending IPX over the WAN requires handling by the processor. There are a limited number of registers to initialize for IPX. The most important registers to configure are the IPXattach-Port register set and the DefaultIPXFlowHandle register. The IPXattach-Port register set is a set of 16 32-bit registers used to store the IPX network address of each attached network.

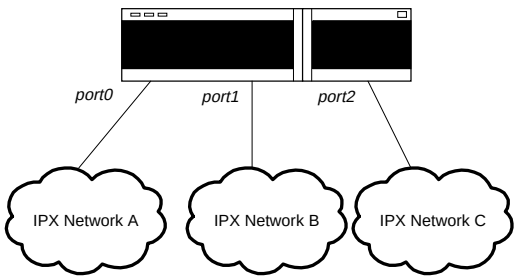


Figure 2-9: IPX Attached Networks

Figure 2-9 shows a router attached to three IPX networks. Each network must have its own IPX network number which is known by the router. The Epoch stores this information internally in the IPXattach-Port Table. In this example the table would have IPXattach-Port0 = A, IPXattach-Port1 = B, and IPXattach-Port2 = C. Upon processing certain IPX packets with source or destination network fields set to 0, the Epoch inserts the IPX network number attached to the network interface the packet was received on.

IPX Queuing

The DefaultIPXFlowHandle register obtains the queue handle used for all received IPX packets. The queue number should be set to whatever priority IPX packets will receive in the networks. Setting the `H bit` (Host Divert bit) results in all IPX packets being sent to the processor port. Setting the `D bit` (Drop bit) similarly filters out all IPX packets.

IPX Forwarding

To properly forward IPX packets, a forwarding table needs to be created in the MUAC RCP. The forwarding entries for IPX have a specific encoding, as shown in Figure 3-22. The forwarding table does not have to be in any particular order, as the Epoch performs a flat address look-up.

LAYER 2 HOST TDM BUS INTERFACE PORT

A conceptual view of the Epoch TDM Bus with a host interface port is shown in Figure 2-10. The precise configuration will depend on the actual customer implementation. The main elements are the TDM Bus, the arbiter and the host interface logic. The bus carries the data packets and control signals to and from the Epoch chip. The arbiter logic determines which of the sixteen TDM Bus interfaces has access to the bus for the current time slot. The TDM Bus host interface logic acts like a normal Layer 2 interface to exchange packets with the switch. Each time slot is divided into two segments – the first one for receiving from the TDM Bus interface to the switch, the second one for transmitting from the switch to the TDM Bus interface logic.

The control lines identify four types of packets to the Epoch switch during a receive operation:

- **IPnormal Mode–**The Layer 2 logic supplies an IPv4 data packet.
- **IPX Mode**–The Layer 2 logic supplies an IPX data packet.
- **Bypass Mode**–The Layer 2 logic supplies a data packet, interface map, and queue number. Entire Layer 2 frames can be switched this way.
- **Reinject Mode**–The Layer 2 logic supplies a data packet, interface map, and queue number. A Layer 3 lookup is performed to provide an L3INDEX for the control bus. Only IP packets may be reinjected.

Furthermore, the Layer 2 interface can abort the Epoch receive process at certain times throughout the transaction. Refer to the *Epoch Data Sheet* for details.

In addition to the packet data, the control lines supply the following information about the packets during a transmit operation:

- **Adjacency Pointer–**Index of Layer 3 destination address in Layer 3/Layer 4 database. Can be used to link Layer 2 and Layer 3 databases.
- **Packet Type–**Same as in the receive side.
- **Queue Number–**Can be used for debugging, Multi-Protocol Label Switching (MLPS), and ATM Virtual Channel (VCI) selection.
- **Layer 4 Index–**Index of Layer 4 Microflow ID in Layer 3/Layer 4 database. Used for system debugging, possibly flow mapping (ATM VPI/VCI), or billing information.
- **Transmitted Byte Count–**The total number of bytes being sent from Epoch. For IP packets, it is not necessarily the same as the length in the header field.
- **Layer 2 Source Interface Port–**Can be used for debugging, encapsulation, and security (IPsec).
- **Buffer Flags–**Indicates whether the packet data storage or packet pointer storage is full or nearly full.
- **Punt Code–**Reason a packet was not processed and is diverted to Port 0. See Table 3-8 for list of Punt Codes.
- **Augment/Divert Options (Snoop, Hbit and Intercept)–**Debugging information. See Figure 3-15 for bit locations.

Packets can be diverted from their normal TDM Bus destination port for the following reasons:

- Source Intercept bit set
- Host Divert bit set
- Packets with a non-zero punt code, for example, a packet with options.

Source intercepted packets are sent to the port indicated by the SIisel register (`0x0e4`), Host diverted packets are sent to the port indicated by the HostForwardPortSelect register (`0x0e8`) if Snoop by Flow is disabled for the normal destination port (See Figure 2-10 below). Punted packet are sent to port zero.

Packets also can be duplicated or snooped. They then are sent to another port in addition to their normal TDM Bus destination port. They can be snooped by destination port, source port or by Microflow on a given destination port by setting the corresponding port bit in DSMAP (`0x0dc`), SSMAP (`0x0e0`) or FSMAP (`0x0f4`) respectively. The packet is forwarded to the normal port and a copy is sent to the snoop port defined by the corresponding register: DSisel (`0x0dc`), SSisel (`0x0e0`), FSisel (`0xf4`). For a flow to be snooped, the `H bit` in its Flow Handle must also be set.

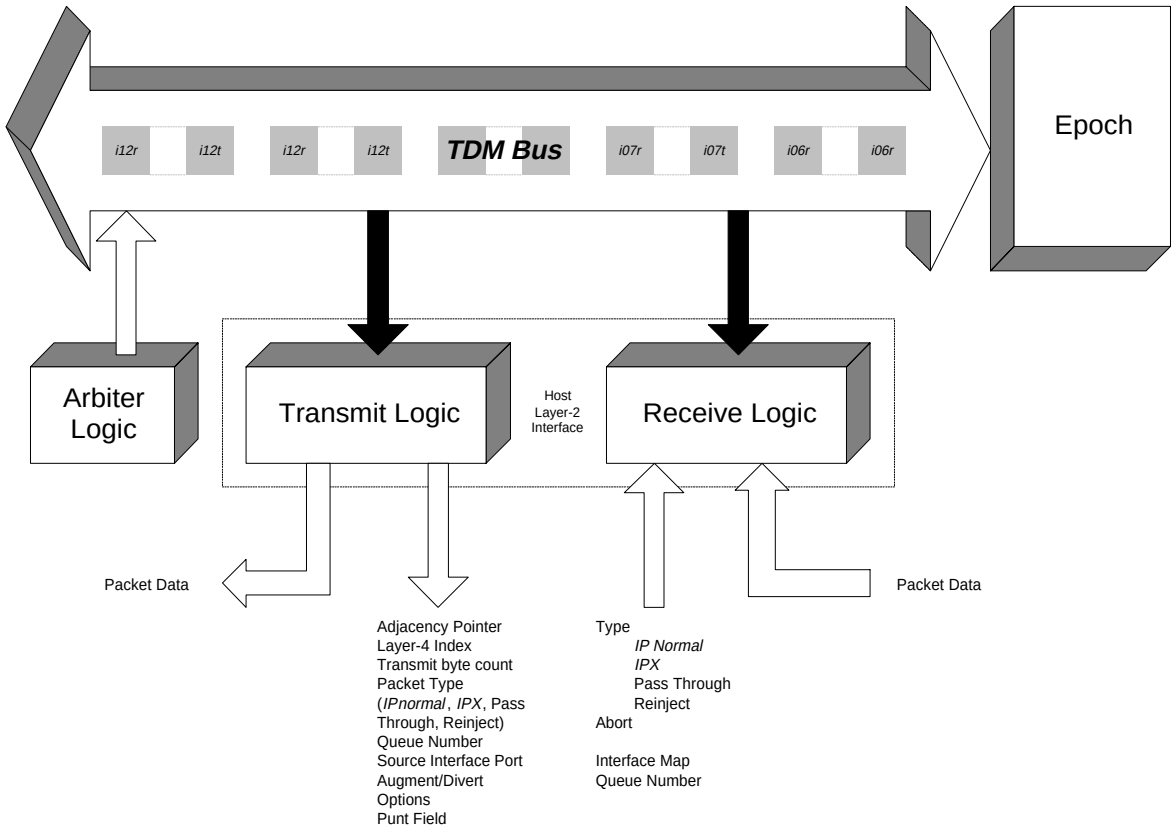


Figure 2-10: Epoch TDM Bus Conceptual View

The majority of packets from the Layer 2 TDM Bus interfaces are forwarded using the TDM Bus interface port map located in the Layer 3 associated data. However, there are several conditions under which packets can be routed to a different interface port. These conditions are summarized in Table 2-2 for IP and Table 2-3 for IPX.

Table 2-2: IP Packets Exception Processing

Feature	Initiated By	Optional Action	Function(s)/Comments
PUNT: Packet Diverted to TDM Bus Port Zero			
Packet Expired	IP header TTL field ≤ 1	DefaultTTLFlowHandle [4] = Drop [2:0] = Queue	RFC0791 standard implementation Punt Code = 0x1
MTU Exceeded	Packet size exceeds MTU of destination port		Host fragmentation; MTU discovery Punt Code = 0x2 Epoch does not fragment packets. Divert only the offending Layer 2 interface destination; all other TDM Bus ports in the portmap bits remain unchanged.
Multicast Source Address	Class D source and destination addresses		Invalid packet filtering Punt Code = 0x3 Not checked for re-injected packets.
Version not IPv4	IP Version Field	DefaultNotVersion4FlowHandle [20] = Drop [18:16] = Queue	Punt Code = 0x4 Epoch processes only IPv4.
Packet with Options	IP header length field greater than five	DefaultIPOptionsFlowHandle [20] = Drop [18:16] = Queue	Punt Code = 0x5 Epoch does not process packets with options.
No Layer 3 Match	IP destination field	No options	Layer 3 filtering, firewall Punt Code = 0x6 Avoids the need to have an all "don't care" entry in the Layer 3 CAM
DIVERT: Packet Diverted to User Selected TDM Bus Port			
Source Intercept	Bit set in SIMAP[15: 0] corresponding to source port to intercept	L4ad [4] = Drop [2:0] = Queue	Simplifies building external firewalls, typically WAN ports connected to the internet. SIIsel[3:0] = Port to divert to. Overrides H bit in flow handle.
SNOOP: Packet Duplicated			
Source Snooping	Bit set in SSMAP[15: 0]	L4ad [2:0] = Queue	Implements RMON functions or Probes. SSMAP[15:0] = Source port to snoop. SSIsel[3:0] = Duplicate packet port.
Destination Snooping	Bit set in DSMAP[15: 0]	L4ad [2:0] = Queue	Implements RMON functions or Probes. DSMAP[15:0] = Source port to snoop. DSIsel[3:0] = Duplicate packet port.
Flow Snooping	Bit set in FSMAP[15: 0]	L4ad [2:0] = Queue	Implements RMON functions or Probes. FSMAP[15:0] = Source port to snoop. FSIsel[3:0] = Duplicate packet port.

Table 2-2: IP Packets Exception Processing (continued)

Feature	Initiated By	Optional Action	Function(s)/Comments
DROP: Packet Dropped			
Multicast TTL Threshold Exceeded	IP header TTL field	L4ad [3] = H bit set: Divert to TDM Bus port zero [2:0] = Queue	Limits the scope of multicast messages. Applies only to multicast messages (class D). Divert only the offending Layer 2 interface destination; all other TDM Bus ports in the portmap bits remain unchanged.
NORMAL: Packet Forwarded to Normal Destination, All Exception Processing is Optional			
Packet Fragmented	IP header MF flag (more fragments)	DefaultFragPackFlowHandle [15:8] = DS replacement field [7] = T8 (replace 8 bit or 6 bit) [6] = R (replace bit) [4] = Drop [3] = Divert to host [2:0] = Queue	Route according to L3ad. Only first packet fragment has TCP/UDP port numbers. Subsequent fragments do not, thus preventing their Layer 4 classification.
Non TCP/UDP Packet	IP protocol field	DefaultUnknownL4ProtoFlowHandle [20] = Drop [19] = Divert to host [18:16] = Queue	Non TCP/UDP packets inspection, routing, and prioritization or dropping. Route according to L3ad. No data for a Layer 4 classification
Layer 4 Classification	Layer 4 match SA-DA-SP- DP-L2SP	L4ad [15:8] = DS replacement field [7] = T8 (replace 8 bit or 6 bit) [6] = R (replace bit) [4] = Drop [3] = Divert to HostForewardPortSelect [2:0] = Queue	Traffic prioritizing, diverting or dropping based upon Layer 4 lookup. Route according to L3ad.
PDQ Classification	No Layer 4 match, SP-DP indexing PDQ table	L4ad [15:8] = DS replacement field [7] = T8 (replace 8 bit or 6 bit) [6] = R (replace bit) [4] = Drop [3] = Divert to HostForewardPortSelect [2:0] = Queue	Traffic prioritizing, diverting or dropping based upon Layer 4 port numbers. Can be used for learning new flows. Route according to L3ad
Other			
Bypass Mode, Layer 2 Bridging	Layer 2 interface asserts parse code `0x00` during control bus receive operation	Portmap and queue provided by Layer 2 interface	Facilitates Layer 2 bridging and handling non-IP traffic (for example, IPX type 20 resent by host). Packets are not processed by MLP. No Adjacency pointer is provided by Epoch.
Reinject	Layer 2 interface asserts parse code `0x01` during control bus receive operation	Route according to L3ad	Used for packets fully processed by the host where no further Epoch processing is needed. Snoop, Intercept, MTU checking, TTL processing and Host Forwarding are ignored. Adjacency pointer is provided by Layer 3 lookup.

Table 2-3: IPX Packets Exception Processing

Feature	Initiated By	Optional Action	Function(s)/Comments
PUNT: Packet Diverted to TDM Bus Port Zero			
Packet Expired	IPX transport control field >=15	DefaultIPXFlowHandle [4] = Drop [2:0] = Queue	IPX standard
Broadcast Packet	IPX packet type field = 20	DefaultIPXFlowHandle [4] = Drop [2:0] = Queue	Netware routing
Invalid Checksum	IPX checksum not `0xffff`	DefaultIPXFlowHandle [4] = Drop [2:0] = Queue	Error handling

Snooping

The snooping feature allows packets to be sent to another TDM Bus port, usually the TDM Bus host interface port, in addition to their normal destination port. There are three snooping capabilities:

- **By Source**–All packets arriving on a given TDM Bus port or ports are copied to another TDM Bus port.
- **By Destination**–All packets destined to a given TDM Bus port or ports are copied to another TDM Bus port.
- **By Flow**–All packets belonging to a given flow and destined to a given TDM Bus port or ports are copied to another TDM Bus port.

Host Divert Feature - H bit

A host bit (`H bit`) resides in each of the six locations shown in Table 2-4. The `H bit` function is to divert packets to the TDM Bus port specified by the HostForwardPortSelect register, typically the host processor, instead of the interface resulting from the Layer 3 lookup. If the normal destination port bit is set in the Snoop by Flow register FSMAP, then the packet is not diverted, but rather snooped, with a copy sent to the port specified by register FSIsel. In general, the host divert feature is provided to all methods to make them consistent with each other. In the case of Layer 3 Associated Data, host diversion is performed by setting the host port bit in the port bit map and resetting all other bits.

Table 2-4: Host Divert Bit Locations

Feature	H Bit Location
Layer 3	Setting of Port Bit Map
Layer 4 flow	Associated Flow Handle
TCP/UDP port default	PDQ Storage entry
Fragmented IP packet	DefaultFragPackFlowHandle register
Non-TCP/UDP IP packet	DefaultUnknownL4ProtoFlowHandle register
IPX packet	DefaultIPXFlowHandle register

Flows present in the Layer 3/Layer 4 database can have the `H bit` in their associated flow handle set so that all packets belonging to that flow are diverted to the port specified by the HostForwardPortSelect register. After receiving the diverted packet, the host can then decide to drop the flow or reinject it. For example, the FTP control port in a flow, well known port number 2110, can have the `H bit` set so that the processor is notified whenever the flow changes any aspect of the FTP connection. This can be used to limit access to certain systems, directories or files.

The `H bit` in a PDQ flow handle allows any TCP/UDP flow not in the Layer 3/Layer 4 database to be optionally sent to the host for further examination. This is an ideal method for notifying the processor that a new flow has been recognized and an entry is needed in the Layer 3/Layer 4 database. Another use is to implement an access authorization process on the host processor for particular applications like FTP (TCP/UDP port numbers 2010 and 2110) or Telnet (TCP/UDP port number 2310).

Non-TCP/UDP packets can be switched to their destination TDM Bus interface port or diverted to the host processor for further analysis by setting the H bit in the DefaultUnknownL4ProtoFlowHandle register. Many non-TCP/UDP packets need to be dropped and not sent out WAN ports (see Figure 1-1 L2port = 10); this can be accomplished with the H bit and host-based analysis software.

Fragmented packets can be examined by the host, for example, to identify and avoid certain denial of service attacks where buffer space is monopolized by the fragmented packets. This is accomplished by setting the H bit in the DefaultFragPackFlowHandle register.

All IPX packets can be examined by the host to insure that the requested services are available at the destination. Furthermore, many IPX packets should not be sent to a WAN port (see Figure 1-1 L2port = 10). By setting the H bit in the DefaultIPXFlowHandle register, IPX packets can be evaluated by host software and either dropped or reinjected.

Drop Packet Feature - D bit

A drop-packet bit (D bit)resides in each of the nine locations shown in Table 2-5. This features is implemented at the Layer 3 level by setting the associated port bit map to all zeros.

Table 2-5: Drop Bit Locations

Feature	H Bit Location
Layer 3	Setting of Port Bit Map
Layer 4 flow	Associated Flow Handle
TCP/UDP port default	PDQ Storage entry
IP packet not Version 4	DefaultNotVersion4FlowHandle register
IP Packet Expired (TTL = 0)	DefaultTTLFlowHandle register
IP packet with options	DefaultIPOptionsFlowHandle register
Fragmented IP packet	DefaultFragPackFlowHandle register
Non-TCP/UDP IP packet	DefaultUnknownL4ProtoFlowHandle register
IPX packet	DefaultIPXFlowHandle register

The D bit in the Layer 4 associated data can be used to prevent flows, deemed malicious, from entering a network. The Layer 4 aging process can delete the entry so that, under different circumstances, the flow can be evaluated again.

PDQ entries can be used to drop packets for a given service. For instance, HTTP traffic, typically occurring on TCP/UDP well known port number 80, can be prevented from entering a system by setting the D bit in the flow handle located in entry 80 of the PDQ storage. Thereafter, any traffic with a TCP/UDP source or destination port of 80 is dropped.

Expired packets (TTL = 0) can be dropped rather than automatically punted to port zero. Non-TCP/UDP packets as well as fragmented packets can be dropped. Dropping fragmented packets can be selected to avoid certain malicious attacks such as: bonk, teardrop and ping of death. Dropping all IPX packets can be important in a network that has no support for the protocol.

Source Intercept

Packets arriving from any Layer 2 TDM Bus interface can be directed to another selected interface port, when there is no Layer 4 match. A bit mapped register, SIMAP, is set to indicate one or more Layer 2 interfaces to be intercepted. A second register, SIIsel, is set with the value of the TDM Bus interface port to which intercepted packets are diverted.

This feature can be used to implement an independent firewall function. As an example, one of the sixteen TDM Bus interface ports can be assigned as a firewall element (see Figure 1-1 L2port = 2 and Figure 1-3). Every packet that arrives on a WAN interface port is then diverted to this port if there is no Layer 4 RCP match; the packet then can be evaluated and either reinjected or discarded. Figure 1-1 shows an example of this where all L2port = 10 (WAN edge) and L2port = 15 (corporate Intranet Edge) traffic is diverted to L2port = 2 for evaluation by a firewall block. Traffic within the department, between nodes at L2port = 4 and L2port = 7, is not sent to the firewall.

Epoch Bypass Mode

The Epoch Bypass feature simplifies using the switch fabric of the chip set for packets that are not processed by Epoch. The burden of determining queuing parameters and the destination Layer 2 TDM Bus interface port map is with the source Layer 2 TDM Bus interface logic. One example of the use of this feature is for IEEE 802.1q VLAN switching. Note that the majority of the logic for this kind of application is with customer supplied logic not contained in the Epoch chip set. The bridging mode is enabled by selecting the Bypass code for the Parse Code, supplying a destination interface portmap, and supplying a queue number to the TDM Control Bus during a receive operation. This is shown in Figure 2-10.

Epoch Reinject Feature

The Reinject feature is available in order for the host processor to process certain packets not handled by Epoch and then forward them to the appropriate Layer 2 TDM Bus interface port using the Epoch switch fabric. As an example, packets that exceed the MTU for a given TDM Bus interface port are not fragmented by Epoch. These packets may be forwarded to the host TDM Bus interface port and the host can fragment the packets and reinject the fragments into the Epoch. Like the Bypass mode, the host TDM Bus interface must supply the interface map and the queue number. However, when the packets are transmitted out of the Epoch, unlike the Bypass mode, the Epoch supplies the adjacency pointer from a Layer 3 lookup. Notice that the Epoch goes no further than supplying the pointer, it does not supply any of the other control data. In the case of a multicast packet where Epoch forwards some packets and others are diverted to the host TDM Bus interface port because of MTU violations, there is additional processing required. The host has to determine which ports to forward to by examining the Layer 3 table and the MTU registers. This information determines how to subdivide the packets for each of the outgoing TDM Bus interfaces.

Initialization and Configuration

HOST PROCESSOR INITIALIZATION

After Power-On-Reset (POR) testing, the Epoch chip and related chips must be initialized prior to system level use. The implementation shown in Figure 1-1 and Figure 1-3 are used throughout this section as an example. Other, customer specific, implementations will modify the software task under discussion. The diagram in Figure 3-1 shows the six storage elements to be initialized. Five of the elements are external to Epoch and one is internal. However, all initialization commands must go through the Epoch processor interface so that access can be coordinated by Epoch's internal logic.

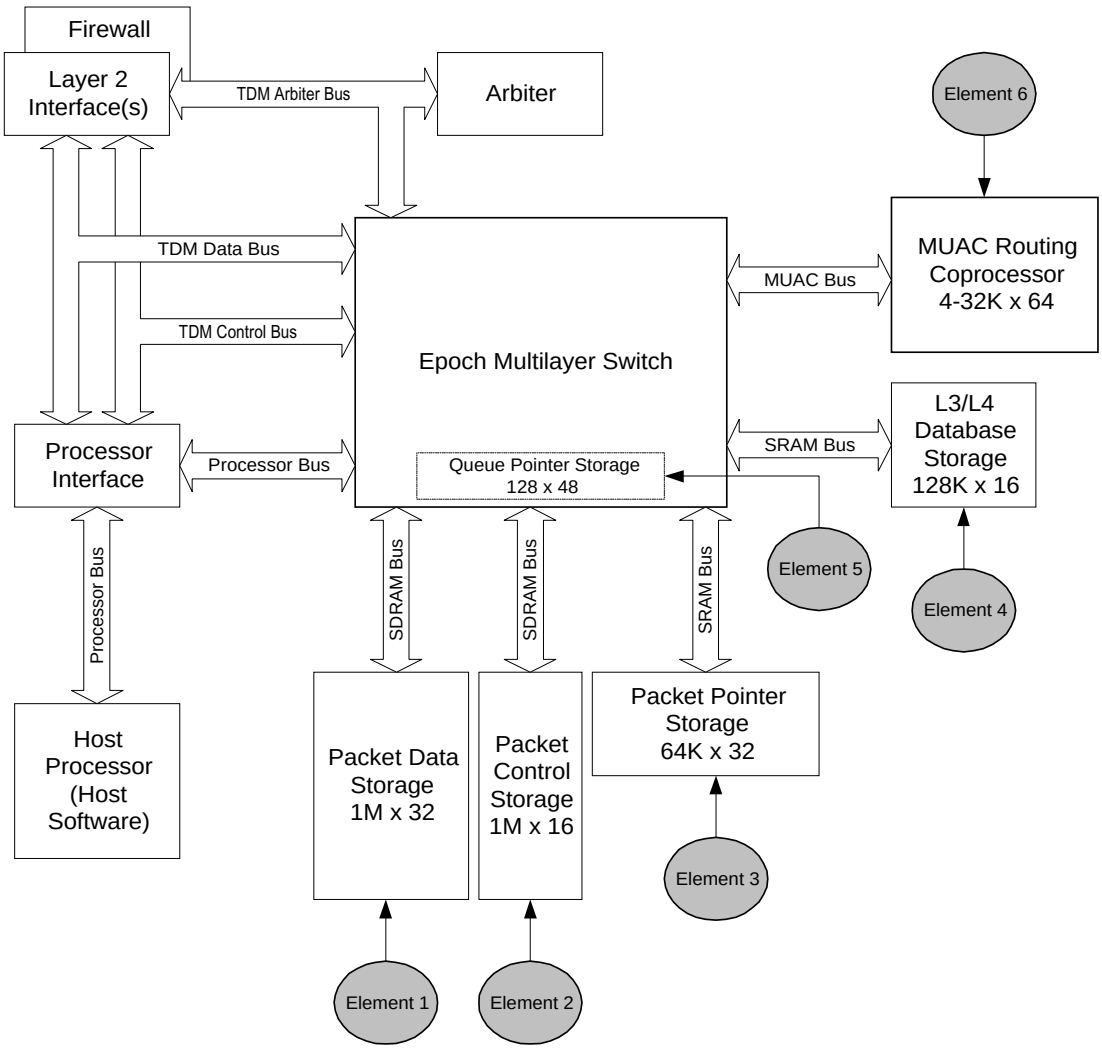


Figure 3-1: Storage Elements to Initialize

The flow chart in Figure 3-2 shows a typical sequence for initializing the Epoch. The twelve steps to initialization contain configuration options unique to a given implementation. The discussions in the following sections assume the most typical implementation.

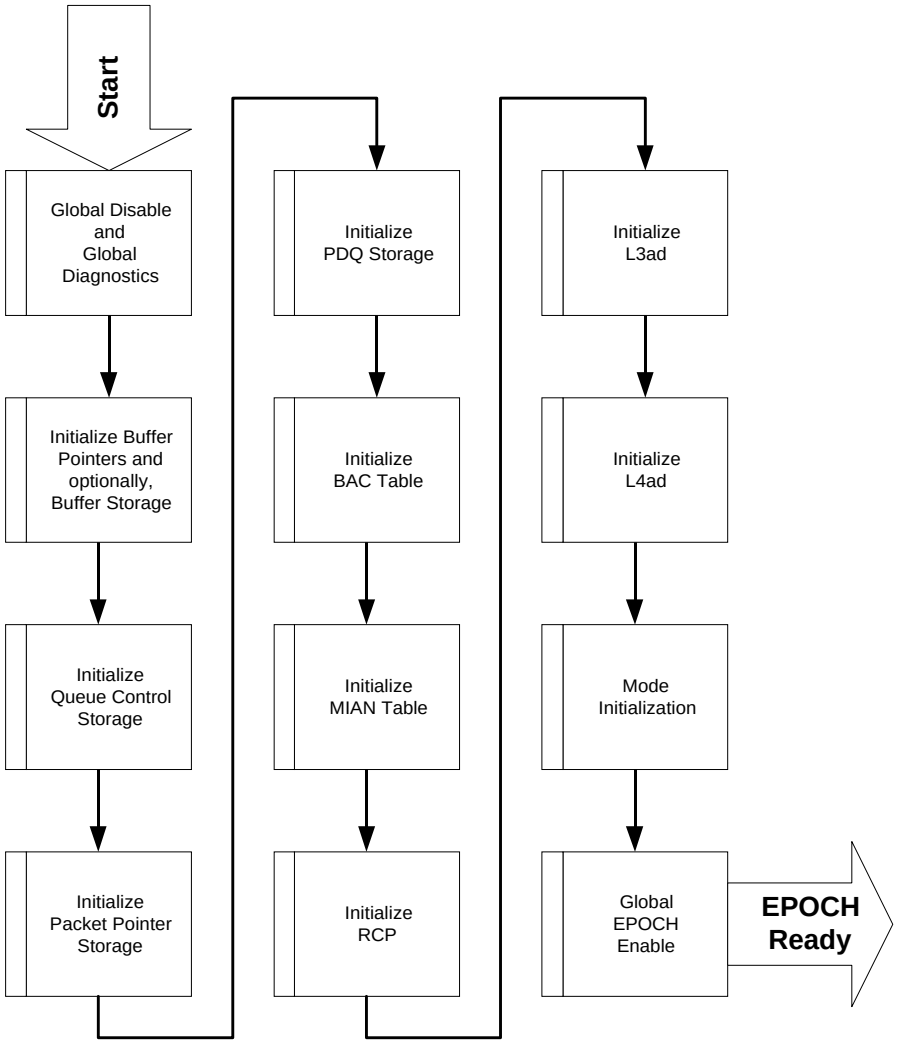


Figure 3-2: Software Initialization Sequence

The code fragment in Figure 3-3 shows routines for reading, writing, and optionally waiting for access to the Epoch processor port. The assumption here is that the hardware design provides a signal that can be read by software, which indicates that Epoch data can be read or written.

All of the code fragments throughout this document have an optional section, controlled by the definition of DIAG_INIT_PHASE, to perform verification tests. For example, when a value is written to some memory location in the Packet Control Storage or Packet Pointer Storage, the DIAG_INIT_PHASE code, if enabled, does a read back and verifies the data was written correctly. Another optional feature is the `Spin_Wait()` routine. This optional routine can be used to optimize access to Epoch internal elements and provide a port ready time-out escape. This prevents the system locking up during a failure.

```
#include address.h
#include oth_code.h
#include epoch_io.h
struct ProcAddrRng SysRng[8] = {                //   for Adr_Chk() and Spin_Wait()
     0x000, 0x05c, "bfm", 0x04,
     0x060, 0x0f4, "pen", 0x01,
     0x108, 0x13c, "cri", 0x40,
     0x140, 0x194, "pkm", 0x08,
     0x198, 0x198, "qsm", 0x02,
     0x19c, 0x1a4, "pim", 0xff,
     0x1a8, 0x1f4, "stm", 0x20,
     0x1f8, 0x1fc, "int", 0x10
};
IOET Host_Write(EPOCH_ADR EpochAddr, EPOCH_DAT Data) { // Module: Host to Epoch
     :
     if(k=Adr_Chk(EpochAddr) > 7)               //   Chk for valid addr
        return(ERR_BAD_ADR);
     :
     return(ERR_NONE);
}
IOET Host_Read(EPOCH_ADR EpochAddr, EPOCH_DAT *Data) { // Module: Epoch to Host
     :
     if(k=Adr_Chk(EpochAddr) > 7)               //   Chk for valid addr
        return(ERR_BAD_ADR);
     :
     return(ERR_NONE);
}
int Adr_Chk(EPOCH_ADR Adr) {                    //   Module: Verify address within
                                                     range
     int i;

     for (i=0; i<8; i++)                        //   Chk each block of addresses
        if (Adr >= SysRng[i].AdrBeg && Adr <= SysRng[i].AdrEnd) break;

     return(i);
}
IOET Spin_Wait(EPOCH_ADR EpochAddr) {           //   Module: Wait for proc port RDY
     :
     if(k=Adr_Chk(EpochAddr) > 7)               //   Chk for valid addr
        return(ERR_BAD_ADR);
     :
     <TBD>
     return(ERR_TYMOUT);                        //   Time-out exit
     :
     return(ERR_NONE);
}
```

Figure 3-3: "C" Functions for Reading and Writing Epoch Internal Registers

Minimal Host-Initiated Diagnostics

As part of the Epoch chip set initialization, it may be important to establish system integrity with diagnostics. All internal RAM, external RAM, external RCP, and many of Epoch's internal functions can be tested from the host processor port.

Note that the TDM Bus is in Sync Suspend mode directly after power-on reset and no bus traffic exists. Diagnostics can be run immediately. However, under other circumstances, the SyncSuspendb register should be set to zero prior to initiating diagnostics, thus preventing TDM Bus traffic from interfering with the test.

RAM Testing

The following four external and one internal RAMs (shown in Figure 3-1) can be tested:

- **Element 1**–Packet Data Storage
- **Element 2**–Packet Control Storage
- **Element 3**–Packet Pointer Storage
- **Element 4**–Layer 3/Layer 4 Database Storage
- **Element 5**–Queue Pointer Storage

These memory elements can be tested in the usual way by simply reading and writing at the appropriate address through the Epoch host port. The testing can be as extensive as required by the application.

Example: *Write with read-back for data integrity tests, write at address, then verify no other location is affected, for addressing tests, and so on.*

RCP Testing

The lookup engine, shown in Figure 3-1 as Element 6 – RCP, can be tested as a memory element and also for search operations. Search operations can be verified by placing data in the CAM at known locations, performing a search and verifying the returned index. This is all controlled by reading and writing registers from the Epoch host port. Again, the level of testing complexity is determined by the needs of the overall product.

Epoch Register Testing

All registers except the following read-only registers can be tested with simple write/read-back tests (see section RAM Testing above). The Epoch should be reset after the completion of such register testing, prior to system use, as a conservative measure.

Table 3-6: Read Only Registers

Register Name	Register Name
BsdramRdData	NextPacket
CntFifoOverflow	NumberofPacket
CntL2Abort	PacketSRAMReadData
CntL3FilterAbort	PortReady
CntL4FilterAbort	ProcessorPortReady
CntOutOfBuffers	PsdramRdData
CntOutOfPacketPointers	QueueSRAMReadDataHigh
CntPacketTooBig	QueueSRAMReadDataLow
CntPacketTooSmall	RCPRdData
CntTotalAborts	RCPSearchResult
CntTTLThreshold	ReadyStatus
HeadofPacket	SRAMRdData
HeadofQueue	TailofQueue
Interrupt_bits	TXPacket
NextFreePointer	Version / ID

Testing Epoch Packet Manipulation Using Only the TDM Bus Host Port

The host processor can send packets to itself, over the TDM Bus, with various options set to test most features of the Epoch. Another way to achieve this is to set the host processor port into Source Intercept mode with the SIMAP and SIIsel registers. The host processor also could set the source intercept register to itself (being careful not to have a Layer 4 entry for the packet) and send TDM Bus packets to any other TDM Bus port. Any of the Host bits, Drop bits, or Touch bits (see Figure 2-2) can be tested without the need for involvement of any other TDM Bus interface. Likewise, the Epoch Diff-Serv features (`T8 bit`, `R bit` and Replacement field) all can be exhaustively tested with this method.

Epoch Register Set

A summary of the register set (in alphabetical order) is shown in Appendix A: Register Summary. Definitions for register addresses and other definitions can be found in Appendix C: C Language Header Files (the registers are listed in address order). The initialization discussed in the following section is the minimum required and is for the most conservative configuration.

Packet Data and Packet Control Storage

The packet data storage contains one or more linked buffers for each packet stream. The packet control storage contains the linked list pointer (head/tail pointers) for these buffers. The relationship between queues, pointers, packets and buffers is depicted in Figure 3-4.

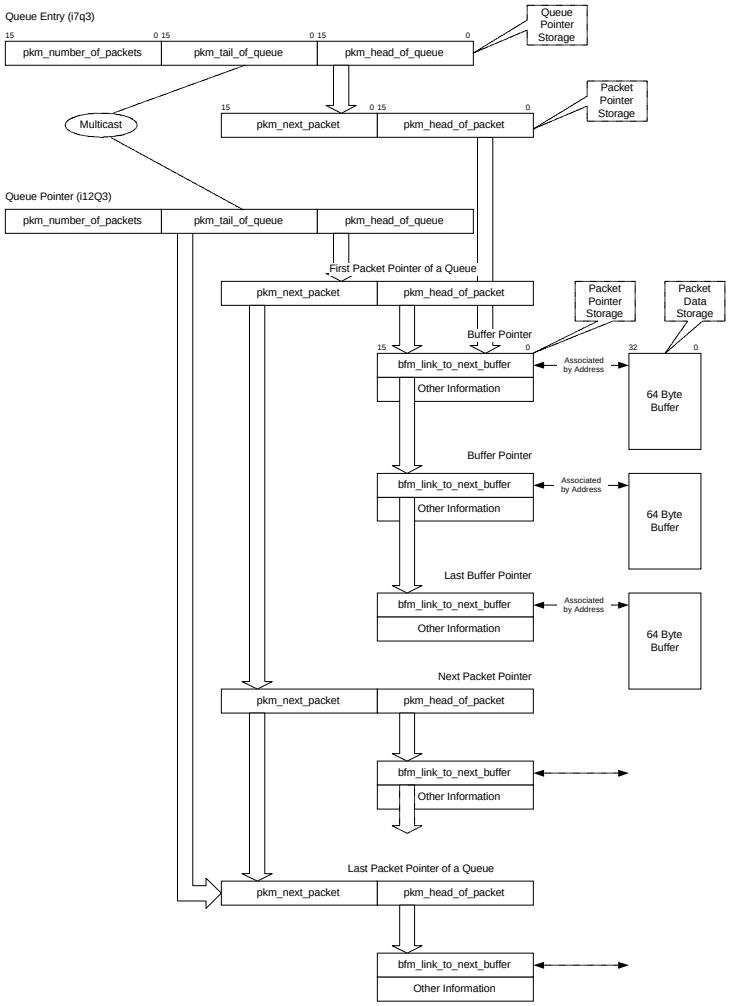


Figure 3-4: Queues, Pointers, Packets, and Buffers

Figure 3-4 also shows the storage location of the various elements. The 48-bit queue pointers are kept in the queue pointer store, the 32-bit packet pointers are maintained in the packet pointer store, the 16-bit buffer pointers are stored in the packet control storage and the 32-bit wide data buffers are saved in the packet data storage. The example given has a multicast packet at queue 3 of TDM Bus interface 7 and interface 12, shown as i7q3 and i12q3 respectively. The head of packet thus points to the same buffer pointer. More details can be found in the *Epoch Data Sheet*.

A code fragment for initializing the buffer pointer memory is given in Figure 3-5. The series of writes before and after the actual writing of the pointer data is required because both the packet data storage and packet control storage are SDRAM based. This code sets the proper RAM access mode for host access and then, normal operation.

```
// Mode setup (burst)
      Host_Write(BSDRAMMRSINIT, 0x0000_0027);
      Host_Write(BSDRAMMRSINIT, 0x0000_0023);
// Mode setup (non-burst)
// Initiate PRECHARGE ALL operation on Buffer SDRAM
      Host_Write(BSDRAMPCALLSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
      Host_Write(BSDRAMRFRSHSTART, 0x00000001);
// Initiate PRECHARGE ALL operation on Buffer Pointer SDRAM
      Host_Write(PSDRAMPCALLSTART, 0x00000001);
// Initiate 8 RESFRESH CYCLES on Buffer Pointer SDRAM
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
      Host_Write(PSDRAMRFRSHSTART, 0x00000001);
// Set Buffer SDRAM MODE to non-burst mode
      Host_Write(BSDRAMMRSINIT, 0x00000020);
// Set Buffer Pointer SDRAM MODE to non-burst mode
      Host_Write(PSDRAMMRSINIT, 0x00000020);
      for (i=0; i<NUMBUFFERS; i++){              //  For each buffer
         Host_Write(BSDRAMWRADDR, i*16);         //  Setup the address
         Host_Write(BSDRAMWRDATA, ((i+1) & 0xffff));//   store pointer
#IFDEF DIAG_INIT_PHASE
         <TBD>                                   //  Do a readback
#ENDIF
      }
// Mode setup (burst)
      Host_Write(BSDRAMMRSINIT, 0x0000_0027);
      Host_Write(BSDRAMMRSINIT, 0x0000_0023);
```

Figure 3-5: Pointer Buffer Memory Initialization

Queue Pointer Storage

Initializing queue pointer storage is shown in Figure 3-6.

```
      Host_Write(ProcessorPortEnable, 1);       //Enable access
      Qptr=0;
      for (i=0; i<16; i++){                     //For each TDM Bus interface
         for (j=0; j<8; j++){                   //For each priority
            Host_Write(QueueSRAMWriteAddress, i<<3 | j); // Setup the addr
            Host_Write(QueueSRAMWriteDataLow, (Qptr<<16 | Qptr);
                                                //Store tail/head pointer
            Host_Write(QueueSRAMWriteDataHigh, 0); // Store packet count
#IFDEF DIAG_INIT_PHASE
            <TBD>                               //Do a readback
#ENDIF
            Qptr++;                             //prepare for next iteration
         }
      }
      Host_Write(ProcessorPortEnable, 0);       //Disable access
```

Figure 3-6: Queue Pointer Storage Initialization

Packet Pointer Storage

Packet pointer storage initialization is illustrated in Figure 3-7.

```
      Host_Write(ProcessorPortEnable, 1);        //  Enable access
      InitSize=NUMBUFFERS;                       //  Buffer count
      for (m=NUMBUFFERS-1; m>=( NUMBUFFERS-InitSize); m--) {
         Host_Write(PacketSRAMWriteAddress, m); // Setup the address
         Host_Write(PacketSRAMWriteData, (m-1) << 16));
                                                 //  write the pointer
#IFDEF DIAG_INIT_PHASE
         <TBD>                                   //  Do a readback
#ENDIF
      }
      Host_Write(ProcessorPortEnable, 0);        //  Disable access
```

Figure 3-7: Packet Pointer Initialization

PDQ Storage

It is suggested to set all PDQ locations (32K by 16 bits) to the value `0x000f` (this bit pattern is illustrated in Figure 3-8)

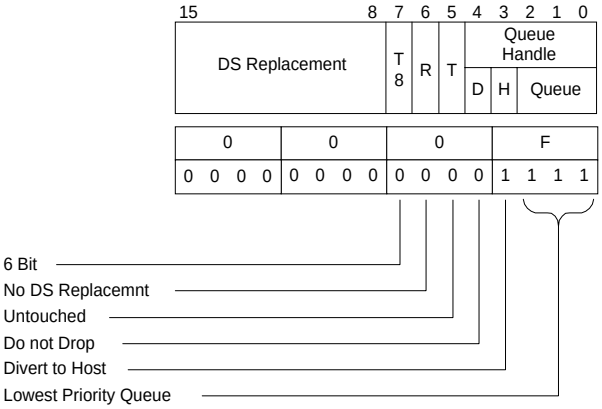


Figure 3-8: Suggested Default PDQ Storage Entry

The default setting mentioned above forces all new flows to be queued to the host TDM Bus interface port so that host software can make a Layer 4 CAM entry if desired. The host could decide to leave flows with this TCP/UDP port number as default flows and simply update the PDQ storage. This would be accomplished by setting the host bit for the PDQ storage entry to zero and then, possibly, changing the priority queue to a higher value.

```
      Host_Write(SRAMWrData, 0x0f);
      for (m=0; m<65536; m++) {                  // For all possible 16-bit port
                                                    values
         Host_Write(SRAMWrAddr, PDQ_STRT + m)
#IFDEF DIAG_INIT_PHASE
         <TBD>                                   // Do a readback
#ENDIF
      }
```

Figure 3-9: PDQ Storage Initialization

BAC Table Initialization

The 256 word deep 16 bit wide BAC table (see Figure 3-17) is stored in 8-bit wide increments (total depth is 512 locations) and starts at location `0x0_8000` and ends at location `0x0_81ff`. Each 16 bit location is coded as shown in Figure 2-2. It is suggested that the table be initialized to `0x0007` as shown in Figure 3-10.

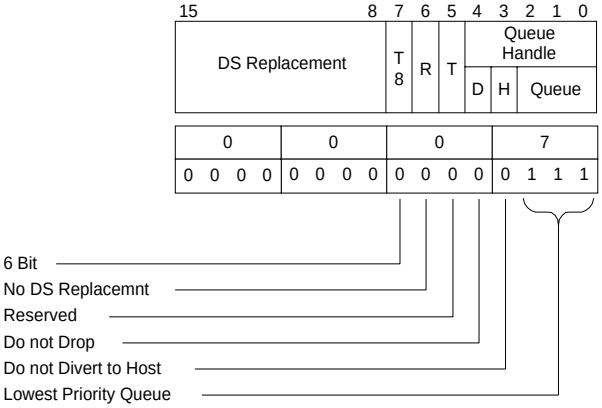


Figure 3-10: Suggested Default BAC Table Entry

```
      for (j=0; j<256; j++) {                        //  Write 0x0007 for every entry
         Host_Read(BAC_MIAN_STRT+(j>>1), &Data);     //  1st byte, Preserve lower byte
         Data &= 0x00ff;
         Host_Write(BAC_MIAN_STRT+(j>>1), Data);
#IFDEF DIAG_INIT_PHASE
         <TBD>                                       //  Do a readback
#ENDIF
         Host_Read(BAC_MIAN_STRT+(j>>1)+1, &Data); //  2nd byte, Preserve lower byte
         Data = (Data & 0x00ff) | 0x0700;
         Host_Write(BAC_MIAN_STRT+(j>>1)+1, Data);
#IFDEF DIAG_INIT_PHASE
         <TBD>                                       //  Do a readback
#ENDIF
      }
```

Figure 3-11: BAC Table Initialization

Another possibility is to initialize the BAC table with the index of the entry (see Figure 3-12). Notice that the replacement bit is zero as a further precaution. Then, whether the BAC mode is enabled or not, there is no change to the DS field on incoming packets.

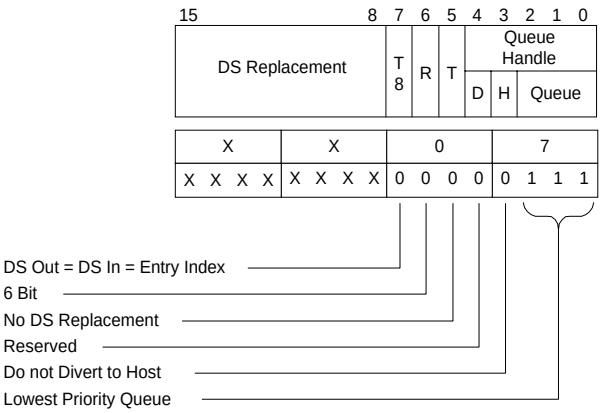


Figure 3-12: Alternate Suggested Default BAC Table Entry

MIAN Table Initialization

The MIAN table is a 4-bit by 32K deep table that is part of the forwarding database (see Figure 3-17). The initialization code fragment is shown below in Figure 3-13. The system configuration being used in this document has the MIAN feature disabled, so initializing this table is optional. Also, when in use, table entries would be set on a per flow basis, as flows are entered into the forwarding database. The code fragment is supplied for an aid to writing diagnostics and to illustrate how table entries are changed.

```
      for (j=0; j<32768; j++) {                  // Write 0 for every entry
         Host_Read(BAC_MIAN_STRT+j, &Data);      // Preserve upper 12-bits
         Data &= 0xfff0;
         Host_Write(BAC_MIAN_STRT+j, Data);
#IFDEF DIAG_INIT_PHASE
         <TBD>                                   // Do a readback
#ENDIF
      }
```

Figure 3-13: MIAN Table Initialization

RCP Initialization

The MUAC RCP (see Figure 3-1 element 6 - RCP) itself has a specific initialization sequence necessary for proper operation. The MUAC configuration register (FR) places the MUAC(s) in the proper mode. The major steps in configuring the MUAC through the Epoch processor interface are as follows:

1. Place all MUAC in hardware mode: set bits FR[27:26] to 002 with a WR FR instruction.
2. Do a WR PA with a unique value on the DQ[31:0] lines. This will write the page address of the highest priority empty MUAC.
3. Set the Full Flag of that MUAC, which cause the next WR PA to address the next lower priority device.
4. Repeat steps 2 and 3 until all the devices have a unique page address.
5. Do a broadcast RST FF instruction to clear all the Full Flags.

Refer to the *MUAC Data Sheet* (see *Related Material*) for more configuration instruction details.

As mentioned earlier, resetting the MUAC RCP sets all memory locations as empty. However, for most applications, it may be useful to set the last location in the CAM to 32 "don't care" trits (ternary digits), Destination = `0x0000` and Mask = `0x0000`. That is, the initialization value is `0x0000_0000` (see Figure 3-18 for details of how to code the value). The Layer 3 associated data value must be initialized as indicated in section Updating the Forwarding Database on page 3-14. This provides a discovery entry so that the host will be notified when packets not in the RCP CAM are encountered. This is not critical as the Epoch forwards all packets that are not in the Layer 3 forwarding database to the host TDM Bus port (the queue number is fixed at seven).

Layer 3/Layer 4 Associated Data Storage

No Layer 3 associated data (see Figure 3-1 element 4 - L3/L4 Database Storage) initialization is necessary as this data must be initialized for every new Layer 3 CAM entry. However, if there is a discovery Layer 3 entry in the CAM (see section RCP Initialization on page 3-10), then the associated data would be a TDM Bus port map for the TDM Bus host. Assuming that the host is TDM Bus port zero (that is, L2port = 0), the value would be `0x0001`.

```
      Host_Write(SRAMWrData, 0x0001);          // L3ad discovery entry
      Host_Write(SRAMWrAddr, 0x0_7fff);        // last location in L3ad
#IFDEF DIAG_INIT_PHASE
      <TBD>                                    // Do a readback
#ENDIF
```

Figure 3-14: Layer 3 Associated Data Initialization

Mode Initialization

The final step in initialization is to configure the Epoch for a specific application. Using the POR default conditions provides a conservative operational mode. This mode includes:

Table 3-7: Operations Default Mode

Feature	Default Mode
Queuing	Round Robin for all ports
TTL processing	Enabled
Multicast Authentication	Disabled
DRAM refresh period	13.36 µs
Non-TCP/UDP flows DefaultUnknownL4ProtoFlowHandle	No H bit, no D bit, queue = 0
Fragmented packet flows DefaultFragPackFlowHandle	No H bit, no D bit, queue = 0
IPX flows DefaultIPXFlowHandle	No H bit, no D bit, queue = 0
Multicast TTL thresholds	All zero: no TTL thresholds
Maximum MTU	Maximum: 0xffff
Source or destination snooping	None
Source interception	None
Differentiated services processing EnableDSclassifier	Disabled
Interrupts Interrupt mask	None

The last step is to enable the device for system level operation by setting the SYNCSUSPENDB register to 1, instructing Epoch to start sending sync pulses to the TDM bus.

LAYER 3/4 SWITCH CONFIGURATION

This section focuses on tasks needed for a complete Layer 3/Layer 4 switch. The *Epoch Reference Design* has complete software and should be consulted for details.

Layer 4 Learning and Aging

The task for the learning and aging process depends on the philosophy chosen as described in section Layer 4 Learning And Aging Methods on page 2-7. The learning task involves adding entries to the RCP as they are discovered (covered in section Updating the Forwarding Database on page 3-14). The aging task involves examining all touch bits (see Figure 2-2) in the Layer 4 associated data (see Figure 3-17) on a regular basis (possibly an examination period of every 10-30 seconds) and either resetting the bit or deleting the entry from the CAM.

Differentiated Services

Differentiated services can be enabled or disable on a per port basis. Once enabled, it can be in microflow mode or behavior aggregate classification mode. Furthermore, there are a number of sub options that can be chosen.

The Six/Eight Bit Field Option

The `T8 bit` set to one means that any DS field replacement will be for the entire eight bit field. If the bit is set to zero, then only the upper six bits will be replaced. The T8 = 0 setting would be used if it is expected that other network elements will use the lower bits. This would be a conservative configuration. The T8 field can be set in four different locations:

- Layer 4 associated data field (see Figure A-1 and Figure 2-2)
- PDQ storage field (see Figure 2-2)
- DefaultIPX_nv4FlowHandles register (see Figure A-1 and Figure 2-2)
- DefaultFrag_UnL4FlowHandles register (see Figure A-1 and Figure 2-2).

BAC Table Setup

The index into the BAC table is the 8 bit DS field from the incoming packet. The data read from the table contains (see Figure 2-2) a replacement DS field, several control bits and the Queue Handle.

Interrupts

Interrupts (see *Epoch Data Sheet* for complete details) occur when one or more of the event counters (see Table 2-1) exceeds its threshold value. Furthermore, there are interrupt conditions related to the condition of buffers. The interrupt bits are reset when the interrupt register is read (they are not reset if the condition persists during the read). There is a mask field and the polarity of the interrupt pin is programmable.

Error Conditions, Exception Handling, and Packet Identifiers

The exceptions, as reported to the Layer 2 TDM Bus interface, are summarized in Figure 3-15 (see *Epoch Data Sheet* for more details).

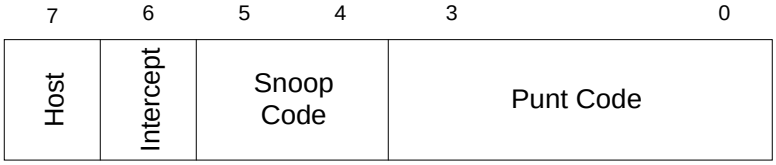


Figure 3-15: TDM Bus Clock 30 Information

The punt codes are defined in Table 3-8 and in the *Epoch Data Sheet*.

Table 3-8: Punt Codes

Mnemonic	Value	Description
NONE	0x0	No punt code
TTL	0x1	The packets TTL field has been decremented to zero.
MTUerr	0x2	The packet needs to be fragmented because it exceeds MTU.
DROPSAM	0x3	This packet was dropped because it has a multicast source address (Class D: 0xEXXX_XXXX).
NOT4	0x4	The packet is IP but not IPv4.
OPT	0x5	This packet has options. The header length is greater than five.
NoL3Match	0x6	No Layer 3 CAM entry was found.
	0x7	Reserved
	0x8	Reserved
	0x9	Reserved
	0xA	Reserved
	0xB	Reserved
	0xC	Reserved
IPXTC	0xD	The IPX packet has a Transport Control field that exceeds 15.
IPXTYPE20	0xE	The IPX packet has a type field of 20, a broadcast packet that needs special handling.
IPXCKSUM	0xF	The IPX packet has a check sum that is not 0xFFFF.

Bad TTL - TTL

This error occurs when a packet arrives with a TTL of 1 or less. In the case of a unicast IP packet, an ICMP message should be generated by the host. In the case of a multicast IP packet, the host would take no action.

MTU Violation - MTUerr

This condition exists when an arriving packet has a size larger than the value specified in the MTU register for that destinations TDM Bus interface port. This condition passes the packet to the host TDM Bus interface for fragmentation. The host fragments the packets and sends each fragment, using pass-through mode, to the destination. The host must examine the port bit map to decide the fragment size and which, of possibly many, TDM Bus interfaces to forward packets.

Invalid IP Address for Multicast - DROPSAM

This error occurs when a source address of a multicast packet is in the multicast range of addresses (that is, if the IP destination address is multicast then the source address can not be multicast).

IP but Not IPv4 Packet - NOT4

This condition is prompted by a packet that is IP but not version 4. Epoch does not handle these packets and they are diverted to the host TDM Bus interface.

Packet with Options - OPT

All packets that have IP options are forwarded to the host for processing. The IP options field is a variable length list of information. It's uses include: security implementations, record route features, source routing and time stamp operations. Packets with options are seldom used and therefore best processed by host software.

No Layer 3 for Unicast - NOL3MATCH

This occurs when the Layer 3 lookup finds no entry. This means that no entry for this IP destination address was placed in the RCP by the host. The host could add the entry into the CAM and update the Layer 3 associated data. Layer 4 manipulations also could be done at this point.

IPX - Excess Transport Control Field - IPXTC

The IPX packet has a transport control field that exceeds 15.

IPX - Broadcast Packet - IPXTYPE20

The IPX packet is a broadcast packet.

IPX - Invalid Checksum - IPXCKSUM

The IPX checksum field is not the value `0xffff`.

Multicasting

The MIAN field in Layer 4 Associated Data and the MIANenb register need setting depending on the topology of the network (see section Multicast TDM Bus Interface Authentication Number - MIAN on page 2-9).

Snooping

The Snooping feature caused the packet to be sent to the host. The host examines the packet for RMON II data capture and update its internal state.

Source Intercept

The packet was diverted to this Layer 2 TDM Bus interface (SIisel register) because the SIMAP register bit was set for that source TDM Bus interface. If this was done for firewall reasons, the packet is examined by the host and the appropriate action taken (drop, forwarded, and/or Layer 4 Associated Data table update).

Host Intercept

There are six methods to divert packets to the host (see section Host Divert Feature - H bit on page 2-17). The type of host processing depends on which method brought it to the host and the philosophy used to set up the condition.

IPX Processing

IPX processing involves routing table maintenance and setting the IPXattach register. Routing tables updates for IPX are the same as any other table update. IPXattach register maintenance is the same as reading/writing any other register.

Non-TCP/IP Protocols

Non-TCP/UDP packets can be forwarded or diverted to the host TDM Bus interface with the DefaultFrag_UnL4FlowHandles register. If they are diverted, they must be treated by the host processor.

Updating the Forwarding Database

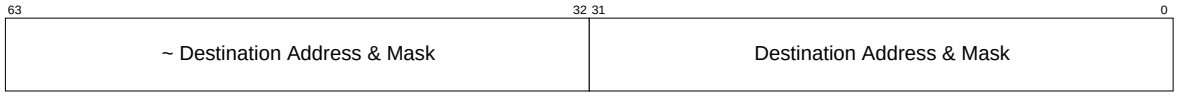
The forwarding database consists of the Layer 3 CAM data (within the RCP device), Layer 4 CAM data (within the RCP device), Layer 3 associated data and the Layer 4 associated data. The Layer 3 CAM data contains ternary unicast IP addresses, binary multicast IP addresses, and binary IPX addresses (see Figure 3-16). The associated data is all held in external storage and the layout is shown in Figure 3-17. The Layer 3 associated data contains TDM Bus interface port maps and multicast MIAN fields. The Layer 4 CAM contains unicast parent and unicast child entries. The Layer 4 associated data contains PDQ storage, Queue number, Host bit, Drop bit, Touch bit, Diff-Serv Replace Request bit, Diff-Serv replacement length selection bit, and Diff-Serv replacement field.

This section details database functions including: formatting entries, creating initial entries, adding new entries, and deleting old entries. All entries for Layer 3 and Layer 4 are held in the same RCP device. Therefore, the entries must be formatted (coded) so that searches for one type of data (for example, ternary search for a Layer 3 entry) only finds that type. The derivation for coding of the entries is beyond the scope of this document but can be found elsewhere. It should be noted that the RCP is a 64-bit wide device that is capable of both 32-bit ternary and 64-bit binary searches. The Layer 4 search is 128-bits wide[1] and thus takes two entries with two searches to complete [2].

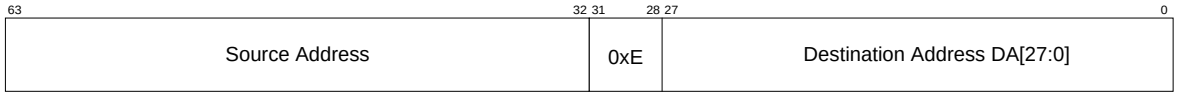
Notes:

1. *A 32-bit IP source address, 32-bit IP destination address, 16-bit TCP/UDP source port, 16-bit TCP/UDP destination port, 4-bit TDM Bus interface source and 28-bit anti-collision field.*
2. *More details can be found in AN-N27 Using MUSIC Devices and RCP's for IP Flow Recognition.*

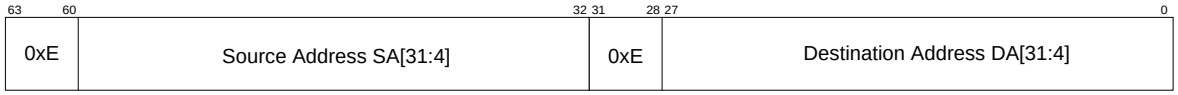
IP Unicast Ternary Structure



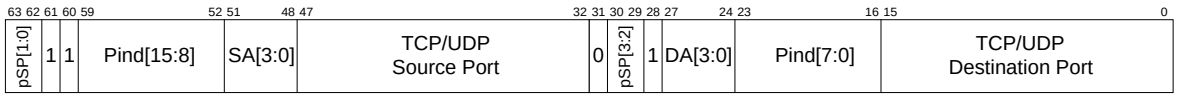
IP Unicast Binary Structure



IP Layer 4 Parent Binary Structure



IP Layer 4 Child Binary Structure



IPX Network Address Binary Structure

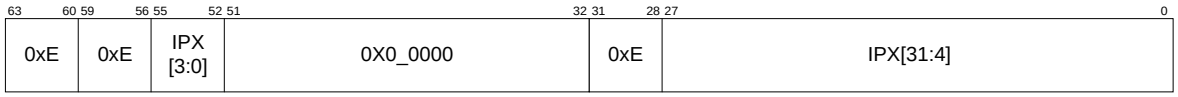


Figure 3-16: RCP CAM Internal Data Layout

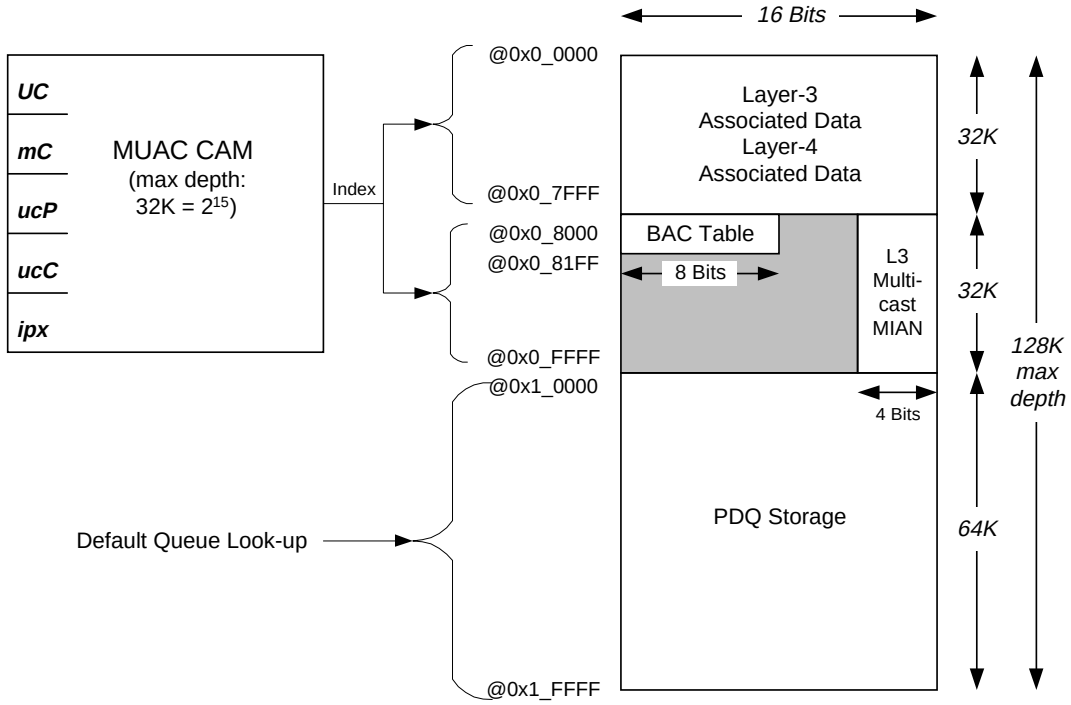


Figure 3-17: Layer-3/4 Database Storage - Internal Data Layout

Layer 3 Information

There are three types of Layer 3 database search entries:

- 32-bit ternary IP unicast
- 64-bit binary IP multicast
- 32-bit binary IPX

The Layer 3 CAM entries for IP unicast packets are ternary values with a "don't care" mask applied. The RCP performs the ternary search on data that is formatted before it is written to the RCP's CAM. The formatting is shown in Figure 3-18 and must be performed by customer logic (either hardware or software). The mask value is the CIDR mask where a zero in any bit position would indicate a "don't care".

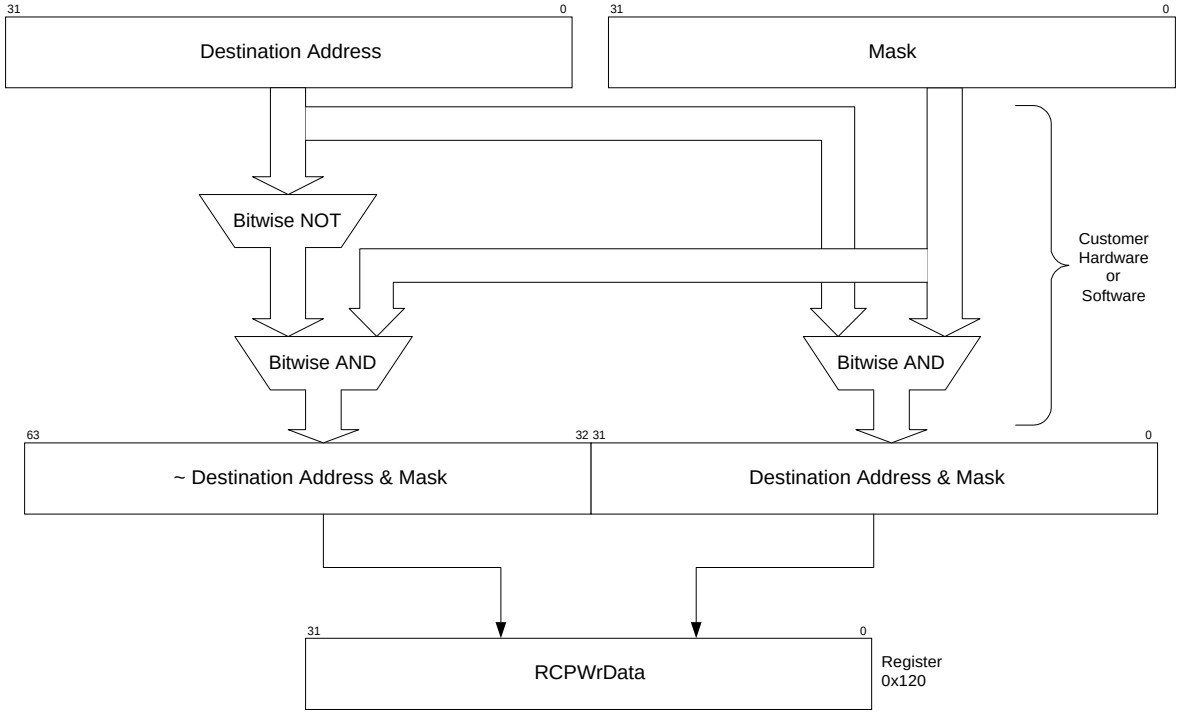


Figure 3-18: Layer-3 CAM IP Unicast Entry - Ternary Search

The TDM Bus interface port bit map is a 16-bit value written to the same 32K index as the RCP's CAM entry. The value is a bit map with position zero corresponding to TDM Bus interface port zero (usually the host Layer 2 port), bit one to TDM Bus interface port one and so on. Figure 3-19 shows the format.

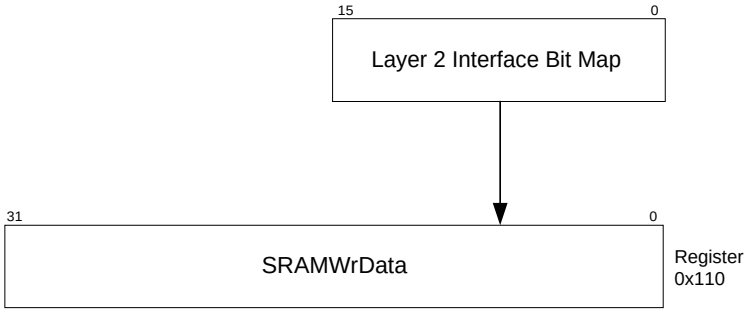


Figure 3-19: Layer 3 Associated Data IP Unicast/IP Multicast Entry

Multicast searches require a 64-bit binary lookup on the 32-bit IP source address and the 32-bit IP destination address as shown in Figure 3-20. The associated data for multicast is the same as unicast and is shown in Figure 3-19.

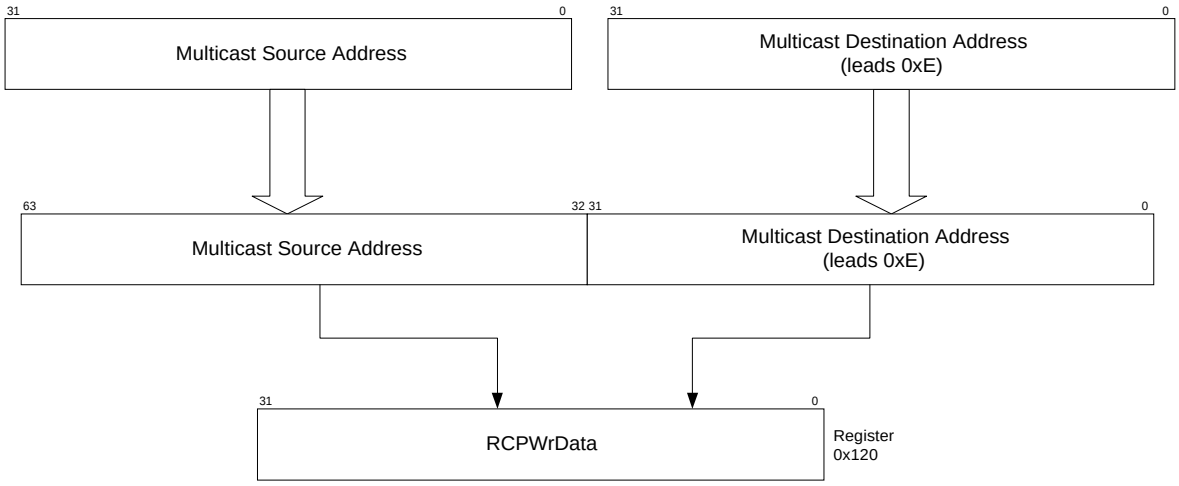


Figure 3-20: Layer 3 CAM IP Multicast Entry - Binary Search

In addition to the TDM Bus port bit map, multicast packets can be dropped if they are not from a specific Layer 2 port. This data is stored in the associated data RAM starting at `0x0_8000`. Note that the upper bits are shared with the BAC table. Therefore, changing these values requires reading the existing value, masking off the lower 4 bits and then ORing in the MIAN value. This is done with customer supplied software or hardware and is shown in Figure 3-21.

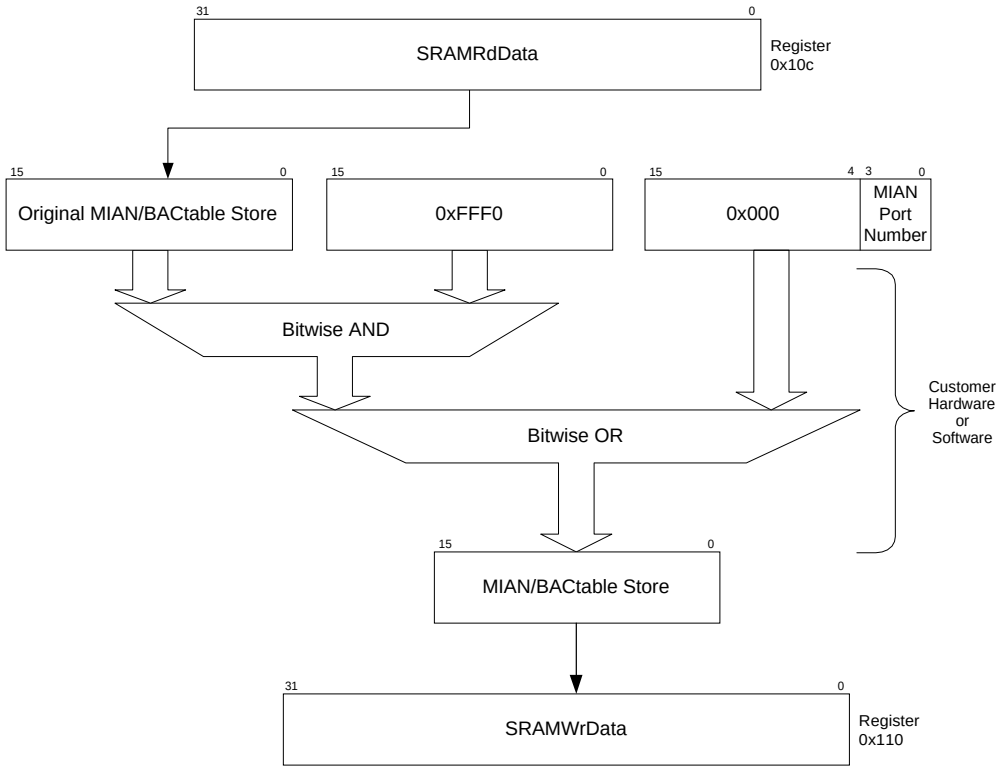


Figure 3-21: MIAN Multicast Entry

The setting of the MIAN field depends on the configuration of the network attached to the Epoch based product. It is updated only when new Layer 3 multicast entries are made. Normally, all Layer 2 TDM Bus interfaces would be enabled for multicast traffic. That is, the CREG0 register (see Appendix A: Register Summary) would have every MIAN bit set to zero. If a configuration exists as in Figure 2-7, for example, where port P1 for router R1 is L2port = 1, then the MIAN entry would be 0x1. This indicates that any multicast packets with a Layer 3 match for this entry would be queued only from L2port = 1 and all others would be dropped.

Layer 3 searches for IPX packets require a 32-bit binary search on the IPX destination network address (see Appendix B: Packet Header Layouts). The value is formatted as shown in Figure 3-22 prior to writing into the RCP.

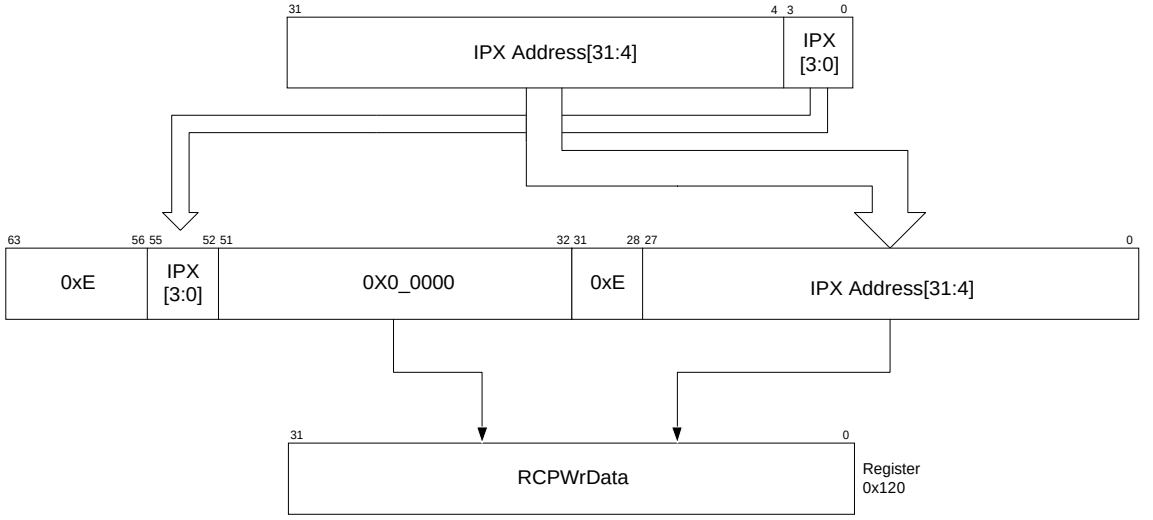


Figure 3-22: Layer 3 CAM IPX Entry - Binary Search

The associated data for IPX is the same as for IP unicast and is shown in Figure 3-19.

Layer 4 Information

The Layer 4 CAM entry is 128-bits wide and thus requires two CAM entries in the RCP and two connected CAM searches. The format of the first search (parent data) is shown in Figure 3-23.

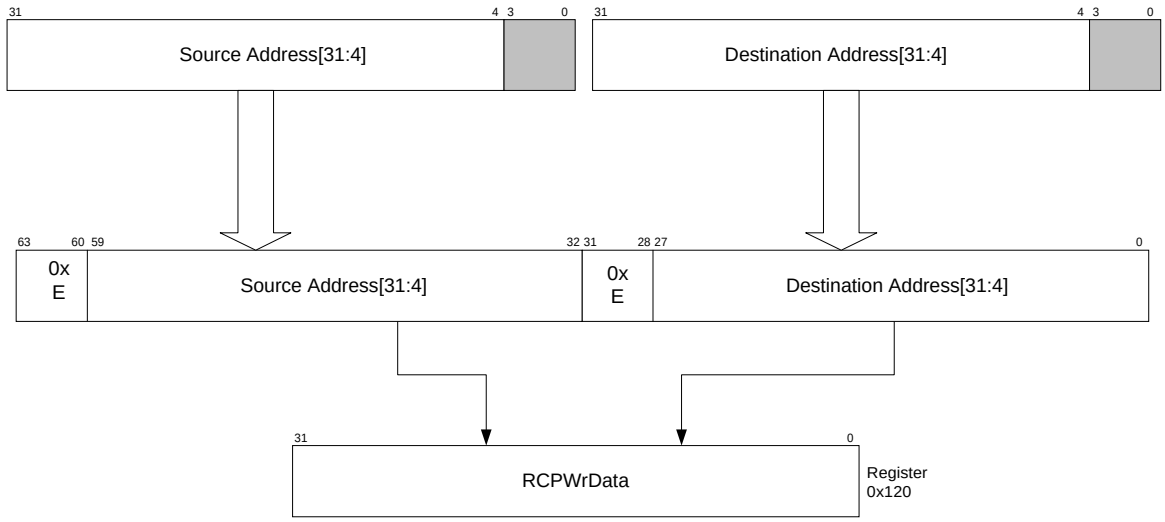


Figure 3-23: Layer 4 Parent CAM Entry

The second part of the Layer 4 search (child data) is shown in Figure 3-24.

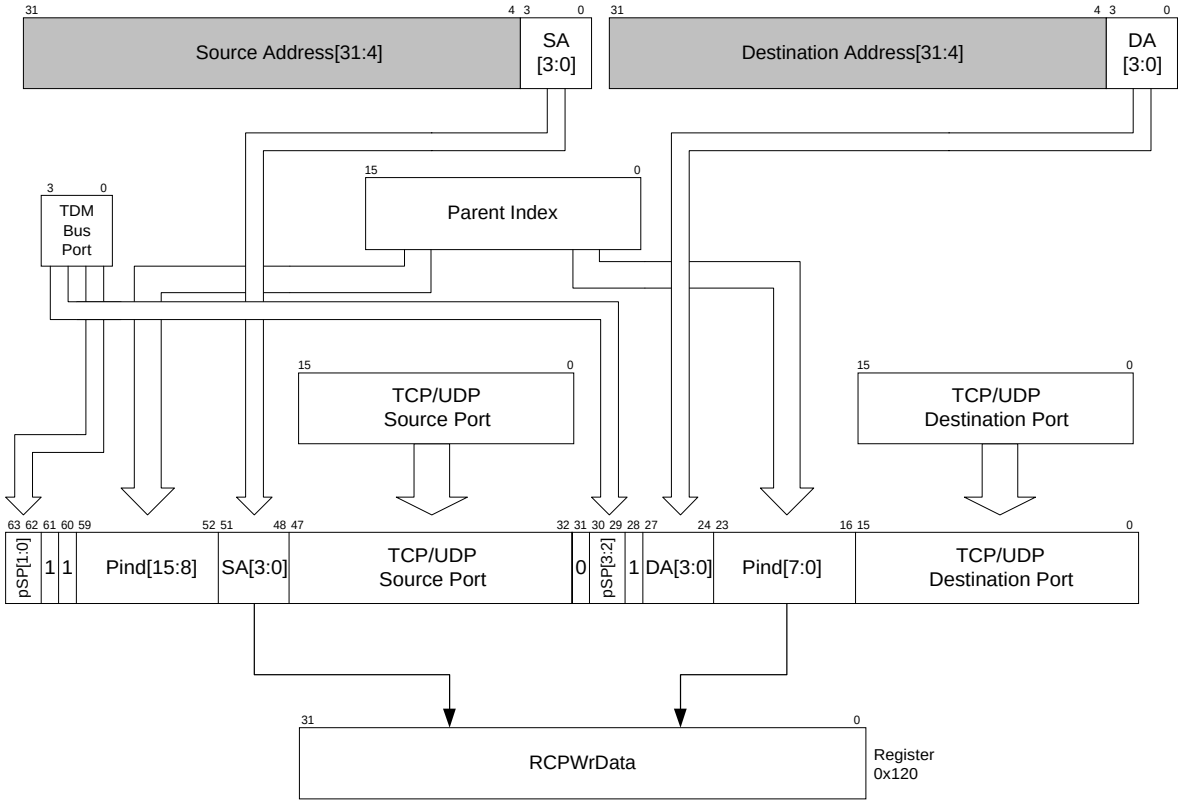


Figure 3-24: Layer 4 Child CAM Entry

The associated data located at the index of the result of the child search is shown in Figure 2-2.

Database Additions and Deletions

The Layer 3 unicast addresses must be put into the CAM in the order of most specific (that is, least number of "don't care" bits in the network mask) first (that is, lowest CAM index). The order of other entries (for example, multicast, IPX, Layer 4 parent or Layer 4 child) is unimportant. It may be necessary to move entries in the CAM so that unicast entries will remain in the proper order. Moving entries will be covered later. Notice that when entries are added to the CAM, the Epoch continues to forward packets. Therefore, care must be taken on how entries are moved and added so as not to cause erroneous database use by Epoch. The correct order is to find an empty CAM location, update the associated data fields at that index, then write the CAM values. In the case of a Layer 4 entry, two free CAM locations are needed. First the associated data is written, then the child entry and last, the parent entry. It is also important to keep in mind that the index of the Layer 4 parent entry is contained in its corresponding child entry (see Figure 3-16).

Appendix A: Register Summary

All the user accessible registers are summarized in this section in alphabetical order for quick reference. Table A-1 shows the registers with field definitions. See the *Epoch Data Sheet* for a list by register address.

Table A-1: Register Summary

Address	Register Name	Access	Bits	Function
0x0ec	BACnMicroflow	R/W	[31:16]	DS mode. DS classification must be enabled. Bit 16 represents port 0, etc. 1 = BAC Mode 0 = Microflow mode
0x028	BsdramMrsInitReg	R/W	[11:0]	Buffer SDRAM mode register setting Resets to 0x020 Set to 0x020 during initialization Set to 0x027 during operation
0x044	BsdramPcAllStart	R/W	[0]	1 = Initiate Precharge All operation on Buffer SDRAM
0x014	BsdramRdAddr	R/W	[19:0]	Processor buffer read address. Access to this register starts the cycle.
0x01c	BsdramRdData	R	[31:0]	Processor buffer read data
0x04c	BsdramRefreshStart	R/W	[0]	1 = Start Refresh to Buffer Memory
0x010	BsdramWrAddr	R/W	[19:0]	Processor buffer write address
0x018	BsdramWrData	R/W	[31:0]	Processor buffer write data. Access to this register starts the cycle.
0x02c	BufFullThreshReg	R/W	[15:0]	BUFFERFULL flag assertion threshold Resets to 0xffff De-asserts when not this value
0x030	BufUsedMaxThreshReg	R/W	[15:0]	BUFFERNEARLYFULL flag assertion threshhold Resets to 0xeb85 (92% full)
0x034	BufUsedMinThreshReg	R/W	[15:0]	BUFFERNEARLYFULL flag de-assertion threshhold Resets to 0xe666 (90% full)
0x1b8	CntFifoOverflow	R	[15:0]	Number of multicast FIFO overflow aborts
0x1bc	CntFifoOverflowThresh	R/W	[15:0]	Interrupt Threshold for above
0x1d0	CntL2Abort	R	[15:0]	Number of aborts from Layer 2 device
0x1d4	CntL2AbortThresh	R/W	[15:0]	Interrupt Threshold for above
0x1e0	CntL3FilterAbort	R	[15:0]	Number of packets dropped because the port bit map is set to 0 in associated data RAM
0x1e4	CntL3FilterAbortThresh	R/W	[15:0]	Interrupt Threshold for above
0x1e8	CntL4FilterAbort	R	[15:0]	Number of Layer 4 filter aborts
0x1ec	CntL4FilterAbortThresh	R/W	[15:0]	Interrupt Threshold for above
0x1a8	CntOutOfBuffers	R	[15:0]	Number of buffer full aborts
0x1ac	CntOutOfBuffersThresh	R/W	[15:0]	Interrupt Threshold for above
0x1b0	CntOutOfPacketPointers	R	[15:0]	Number of out-of-packet-pointer aborts
0x1b4	CntOutOfPacketPointersThresh	R/W	[15:0]	Interrupt Threshold for above
0x1c0	CntPacketTooBig	R	[15:0]	Number of oversize packet aborts
0x1c4	CntPacketTooBigThresh	R/W	[15:0]	Interrupt Threshold for above
0x1c8	CntPacketTooSmall	R	[15:0]	Number of undersized packet aborts IP and IP re-inject packets < 20 bytes IPX packets < 30 bytes
0x1cc	CntPacketTooSmallThresh	R/W	[15:0]	Interrupt threshold for above
0x1f0	CntTotalAborts	R	[15:0]	Total number of packets aborted
0x1f4	CntTotalAbortsThresh	R/W	[15:0]	Interrupt threshold for above
0x1d8	CntTTLThreshold	R	[15:0]	Number of TTL threshold aborts

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x1dc	CntTTLThresholdThresh	R/W	[15:0]	Interrupt threshold for above
0x138	CREG0	R/W	[15:0]	1 = Enable MIAN. Each bit represents a source port, bit 0 is port 0 etc.
0x0d8	DefaultFragPackFlowHandle	R/W	[15:0]	Default flow handle for Fragmented packets
			[15:8]	Replacement DS field
			[7]	T8 (Replace 8 bit or 6 bit) 1 = Replace 8 bits of DS 0 = Replace top 6 bits of DS
			[6]	R (Replace bit) 1 = Replace DS field 0 = Do not replace DS field
			[5]	Not Implemented
			[4]	Drop
			[3]	Divert to host
			[2:0]	Queue Number
0x0d4	DefaultIPOptionsFlowHandle	R/W	[31:16]	Default flow handle for IP Options packets
			[31:21]	Not implemented
			[20]	Drop
			[19]	Not Implemented
			[18:16]	Queue Number
0x0d0	DefaultIPXFlowHandle	R/W	[15:0]	Default Flow handle for punted IPX packets
			[15:5]	Not implemented
			[4]	Drop
			[3]	Divert to Host
			[2:0]	Queue Number
0x0d0	DefaultNotVersion4FlowHandle	R/W	[31:16]	Default flow handle for Not IPv4 Packets
			[31:21]	Not implemented
			[20]	Drop
			[19]	Not implemented
			[18:16]	Queue Number
0x0d4	DefaultTTLFlowHandle	R/W	[15:0]	Default flow handle for TTL = 0 Packets
			[15:5]	Not implemented
			[4]	Drop
			[3]	Not Implemented
			[2:0]	Queue Number
0x0d8	DefaultUnknownL4ProtoFlowHandle	R/W	[31:16]	Default flow handle for unknown Layer 4 protocol packets
			[31:24]	Replacement DS field
			[23]	T8 (Replace 8 bit or 6 bit) 1 = Replace 8 bits of DS 0 = Replace top 6 bits of DS
			[22]	R (Replace bit) 1 = Replace DS field 0 = Do not replace DS field
			[21]	Not Implemented
			[20]	Drop
			[19]	Divert to Host
			[18:16]	Queue Number
0x0e8	DoNotProcess	R/W	[15:0]	Disable TTL decrement per port. Bit 0 is port 0, etc. 0 = Enable
0x0dc	DSisel	W	[19:16]	Destination snoop interface select for augmenting
0x0dc	DSMAP	W	[15:0]	Destination snoop map. Each bit represents the destination port to snoop, for example bit 0 = port 0.

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x0e0	DSisel	R	[19:16]	Destination snoop interface select for augmenting
0x0e0	DSMAP	R	[15:0]	Destination snoop map. Each bit represents the destination port to snoop, for example bit 0 = port 0.
0x0ec	EnableDSclassifier	R/W	[15:0]	Enable DS classification. Each bit represents the port to enable. Bit 0 = port 0, etc. If DS classification is off, flows are classified by microflow but no DS re-marking can occur whether the R bit is set in the flow handle or not.
0x18c	FullThreshold	R/W	[15:0]	Used packet pointer threshold at which BUFFERFULL should be asserted
0x178	HeadofPacket	R	[15:0]	Information: current pkm_head_of_packet field.
0x17c	HeadofQueue	R	[15:0]	Information. Current pkm_head_of_queue field.
0x0e8	HostForwardPortSelect	R/W	[31:28]	Host port number for all packets forwarded or punted to host.
0x1f8	Interrupt bits	R		Each bit is set accordingly when the interrupt condition occurs. The bits are sticky, that is, they remain set until the processor reads this register, when they are automatically cleared at the end of the read operation. The INT bit is asserted when at least one enabled interrupt condition occurs. The INT bit is cleared when this register is read.
			[0]	Buffer full. The main packet buffer is completely full.
			[1]	Buffer nearly full. The main packet buffer's number of used buffers has crossed the upper nearly full threshold.
			[2]	Packet Pointer RAM full. The packet pointer RAM is completely full.
			[3]	Packet Pointer RAM nearly full. The packet pointer RAM has crossed the upper nearly full threshold.
			[4]	Read 0
			[5]	CntOutOfBuffers counter crossed threshold
			[6]	CntOutOfPacketPointers counter crossed threshold
			[7]	CntFifoOverflow filter counter crossed threshold
			[8]	CntPacketTooBig counter crossed threshold
			[9]	CntPacketTooSmall counter crossed threshold
			[10]	CntL2Abort counter crossed threshold.
			[11]	CntTTLThreshold counter crossed threshold
			[12]	CntL3FilterAbort counter crossed threshold
			[13]	CntL4FilterAbort counter crossed threshold
			[15:14]	Read 0
0x1fc	Interrupt mask and INT pin polarity	R/W	[13:0]	Mask for the Interrupt register. If a bit is set the corresponding interrupt is enabled. Resets to 0.
			[14]	Read 0
			[15]	Polarity of INT pin. 0 = Active LOW 1 = Active HIGH.
0x090	IPXattach-Port0	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x094	IPXattach-Port1	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x098	IPXattach-Port2	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x09c	IPXattach-Port3	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0a0	IPXattach-Port4	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0a4	IPXattach-Port5	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0a8	IPXattach-Port6	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0ac	IPXattach-Port7	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0b0	IPXattach-Port8	R/W	[31:0]	IPX Network number per port for IPX forwarding

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x0b4	IPXattach-Port9	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0b8	IPXattach-Port10	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0bc	IPXattach-Port11	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0c0	IPXattach-Port12	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0c4	IPXattach-Port13	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0c8	IPXattach-Port14	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0cc	IPXattach-Port15	R/W	[31:0]	IPX Network number per port for IPX forwarding
0x0f0	L3DefaultIndexPtr	R/W	[14:0]	When no L3 match is obtained, the L3 Index pointer is set to this value.
0x0f0	L4DefaultIndexPtr	R/W	[30:16]	When no Layer 4 match is obtained the Layer 4 Index pointer is set to this value.
0x194	LowerThreshold	R/W	[15:0]	Used packet pointer threshold at which BUFFERNEARLYFULL should be de-asserted.
0x070	MTU-Port0	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port1	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x074	MTU-Port2	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port3	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x078	MTU-Port4	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port5	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x07c	MTU-Port6	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port7	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x080	MTU-Port8	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port9	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x084	MTU-Port10	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port11	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x088	MTU-Port12	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port13	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x08c	MTU-Port14	R/W	[15:0]	Maximum MTU per port for Punting over MTU packets to host.
	MTU-Port15	R/W	[31:16]	Maximum MTU per port for Punting over MTU packets to host.
0x170	NextFreePointer	R	[15:0]	Information: Next Free Packet pointer address.
0x174	NextPacket	R	[15:0]	Information: Next packet to operate on.
0x0e8	NoL3MatchHostForward	R/W	[27]	Forward packets to host port if no L3 match is obtained. 1 = Forward 0 = Drop
0x184	NumberofPacket	R	[15:0]	Information: current pkm_number_of_packets field.
0x160	PacketSRAMReadAddress	R/W	[15:0]	Address of Packet pointer SRAM to perform read. Access to this register starts cycle.
0x164	PacketSRAMReadData	R	[31:0]	Packet pointer read data

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x158	PacketSRAMWriteAddress	R/W	[15:0]	Address of Packet pointer SRAM to perform write
0x15c	PacketSRAMWriteData	R/W	[31:0]	Packet pointer Write data. Access to this register starts cycle.
0x198	PortAlgorithmSelect	R/W		0 = Weighted Round Robin Mode 1 = Priority Mode
			[0]	Port 0
			[1]	Port 1
			[2]	Port 2
			[3]	Port 3
			[4]	Port 4
			[5]	Port 5
			[6]	Port 6
			[7]	Port 7
			[8]	Port 8
			[9]	Port 9
			[10]	Port 10
			[11]	Port 11
			[12]	Port 12
			[13]	Port 13
			[14]	Port 14
			[15]	Port 15
			[31:16]	Reserved. Set to 0x0000.
0x020	PortReady	R	[0]	Ready bit. Indicates when the BFM registers are ready. ***Note:*** *For information only. It is not necessary to poll this bit as register accesses are controlled by the PREADY signal on the processor port.*
0x168	ProcessorPortEnable	R/W	[0]	1 = Enable processor access to BFM and PKM memories 0 = Disable Resets to 0 Must be set to 1 for processor access to PKM memories.
0x16c	ProcessorPortReady	R	[0]	Indicates the processor interface to the PKM is ready. For information only. The PREADY signal controls access. The processor does not need to poll this bit.
0x054	ProgSyncReg	R/W	[5:0]	Number of clocks before the first word of TDM RX Data SYNC pulse is asserted. Resets to 3 (CLK 0).
0x024	PsdramMrsInitReg	R/W	[11:0]	Buffer Control SDRAM mode register settings. Resets to 0x020 Set to 0x020 during initialization Set to 0x023 during operation
0x048	PsdramPcAllStart	R/W	[0]	1 = Initiate Precharge All operation on Buffer Control SDRAM
0x004	PsdramRdAddr	R/W	[19:0]	Processor buffer control read address. Access to this register starts the cycle.
0x00c	PsdramRdData	R	[15:0]	Processor buffer control read data
0x050	PsdramRefreshStart	R/W	[0]	1=Start Refresh to Buffer Control Memory
0x000	PsdramWrAddr	R/W	[19:0]	Processor buffer control write address
0x008	PsdramWrData	R/W	[15:0]	Processor buffer control write data. Access to this register starts the cycle.
0x14c	QueueSRAMReadAddress	R/W	[6:0]	Address of Queue pointer SRAM to perform read. Access to this register starts cycle.
0x154	QueueSRAMReadDataHigh	R	[15:0]	HIGH word of read data
0x150	QueueSRAMReadDataLow	R	[31:0]	LOW word of read data
0x140	QueueSRAMWriteAddress	R/W	[6:0]	Address of Queue pointer SRAM to perform write

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x148	QueueSRAMWritreDataHigh	R/W	[15:0]	HIGH word of write data. Access to this register starts cycle.
0x144	QueueSRAMWriteDataLow	R/W	[31:0]	LOW word of write data
0x11c	RCPRdData	R	[31:0]	CRI RCP Read Data
0x118	RCPRdOp	R/W		CRI RCP Read Operand, launches the read.
			[12:0]	RCP Op-Code (See MUAC RCP Data Sheet)
			[13]	PCPCS20b pin 0 = Set LOW 1 = Set HIGH
			[14]	RCPCS21b pin
			[15]	RCPCS22b pin
			[16]	RCPCS23b pin
			[17]	RCP DSC pin
			[18]	RCP Avb pin
0x12c	RCPSearchData	R/W	[31:0]	CRI RCP Search Data
0x134	RCPSearchIndex	R	[14:0]	CRI RCP Search Index returned from search operation
0x134	RCPSearchMF	R	[15]	CRI RCP Search Match Flag
0x134	RCPSearchMM	R	[16]	CRI RCP Search Multiple Match Flag
0x130	RCPSearchOp	R/W		CRI RCP Search Operand, launches the search.
			[12:0]	RCP OpCode
			[13]	PCPCS20b pin
			[14]	RCPCS21b pin
			[15]	RCPCS22b pin
			[16]	RCPCS23b pin
			[17]	RCP DSC pin
			[18]	Fast Write Compare 0 = Search using RCPSearchData and RCPSearchOp registers 1 = Search using RCPWrData, RCPSearchData, and RCPSearchOP registers
0x128	RCPStatus	R/W	[0]	RCP validity bit. Valid from reads.
			[1]	RCP Full Flag. Valid from writes.
0x120	RCPWrData	R/W	[31:0]	CRI RCP Write Data
0x124	RCPWrOp	R/W		CRI RCP Write Operand, launches the write.
			[12:0]	RCP OpCode
			[13]	PCPCS20b pin
			[14]	RCPCS21b pin
			[15]	RCPCS22b pin
			[16]	RCPCS23b pin
			[17]	RCP DSC pin
			[18]	RCP Avb pin
			[19]	RCP VBb pin
0x1a0	Ready Status. Information only.	R	[0]	PEN
			[1]	QSM
			[2]	BFM
			[3]	PKM
			[4]	INT
			[5]	STM
			[6]	CRI
0x13c	Reserved			
0x058	Reserved			
0x05c	Reserved			

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x19c	Reset/Status	R/W	[5:0]	Software Reset and PLL control
		R/W	[0]	Software reset Write a 1 to reset part Read 0
		R	[1]	PLL Enabled 0 = Disabled 1 = Enabled
			[2]	PLL Locked 0 = Not locked 1 = Locked
		R/W	[4:3]	PLL Multiplier. PLL must run between 100 and 200 MHz. External clock frequency range is indicated in parenthesis. 00: Multiply by 2 (50 to 66 MHz) 01: Multiply by 4 (25 to 50 MHz) 10: Multiply by 8 (12.5 to 25 MHz) 11: Multiply by 16 (6.25 to 12.5 MHz)
			[5]	Clock on INT. Enables system clock to be output on the INT pin for testing purposes. 0 = INT normal (processor Interrupt) 1 = INT is internal clock
0x038	RfshMaxCntReg	R/W	[15:0]	Refresh period. Number of CLK periods between each Buffer and Buffer control SDRAM Refresh cycle. Resets to 0x380 13.58 uS with 66MHz CLK.
0x040	RfshSuspendb	R/W	[0]	0 = Suspend Refresh 1 = Enable Refresh Resets to 0
0x0e4	SIisel	R/W	[19:16]	Source Intercept Interface select for diverting.
0x0e4	SIMAP	R/W	[15:0]	Source Intercept map. Each bit represents the source port to intercept, for example bit 0 = port 0.
0x108	SRAMRdAddr	R/W	[16:0]	CRI SRAM read address, launches the read.
0x10c	SRAMRdData	R	[15:0]	CRI SRAM read Data
0x114	SRAMWrAddr	R/W	[15:0]	CRI SRAM Write Address, launches the write.
0x110	SRAMWrData	R/W	[16:0]	CRI SRAM Write Data
0x0dc	SSisel	R	[19:16]	Source Snoop Interface select for augmenting.
0x0dc	SSMAP	R	[15:0]	Source Snoop map. Each bit represents the source port to snoop, for example bit 0 = port 0.
0x0e0	SSisel	W	[19:16]	Source Snoop Interface select for augmenting.
0x0e0	SSMAP	W	[15:0]	Source Snoop map. Each bit represents the source port to snoop, for example bit 0 = port 0.
0x03c	SyncSuspendb	R/W	[0]	0 = Suspend SYNC pulse 1 = Enable SYNC pulse Resets to 0
0x180	TailofQueue	R	[15:0]	Information: current pkm_tail_of_queue field.
0x060	TTLThreshold-Port0	R/W	[7:0]	TTL Threshold per port for Multicast TTL Thresholding
0x060	TTLThreshold-Port1	R/W	[15:8]	TTL Threshold per port for Multicast TTL Thresholding
0x060	TTLThreshold-Port2	R/W	[23:16]	TTL Threshold per port for Multicast TTL Thresholding
0x060	TTLThreshold-Port3	R/W	[31:24]	TTL Threshold per port for Multicast TTL Thresholding
0x064	TTLThreshold-Port4	R/W	[7:0]	TTL Threshold per port for Multicast TTL Thresholding
0x064	TTLThreshold-Port5	R/W	[15:8]	TTL Threshold per port for Multicast TTL Thresholding
0x064	TTLThreshold-Port6	R/W	[23:16]	TTL Threshold per port for Multicast TTL Thresholding
0x064	TTLThreshold-Port7	R/W	[31:24]	TTL Threshold per port for Multicast TTL Thresholding
0x068	TTLThreshold-Port8	R/W	[7:0]	TTL Threshold per port for Multicast TTL Thresholding
0x068	TTLThreshold-Port9	R/W	[15:8]	TTL Threshold per port for Multicast TTL Thresholding

Table A-1: Register Summary (continued)

Address	Register Name	Access	Bits	Function
0x068	TTLThreshold-Port10	R/W	[23:16]	TTL Threshold per port for Multicast TTL Thresholding
0x068	TTLThreshold-Port11	R/W	[31:24]	TTL Threshold per port for Multicast TTL Thresholding
0x06c	TTLThreshold-Port12	R/W	[7:0]	TTL Threshold per port for Multicast TTL Thresholding
0x06c	TTLThreshold-Port13	R/W	[15:8]	TTL Threshold per port for Multicast TTL Thresholding
0x06c	TTLThreshold-Port14	R/W	[23:16]	TTL Threshold per port for Multicast TTL Thresholding
0x06c	TTLThreshold-Port15	R/W	[31:24]	TTL Threshold per port for Multicast TTL Thresholding
0x188	TXPacket	R	[15:0]	1 = A packet is being transmitted b0 = port0, bit1 = port1, etc.
0x190	UpperThreshold	R/W	[15:0]	Used packet pointer threshold at which BUFFERNEARLYFULL should be asserted.
0x1a4	Version / ID	R	[7:0]	Revision
			[23:8]	0xec01

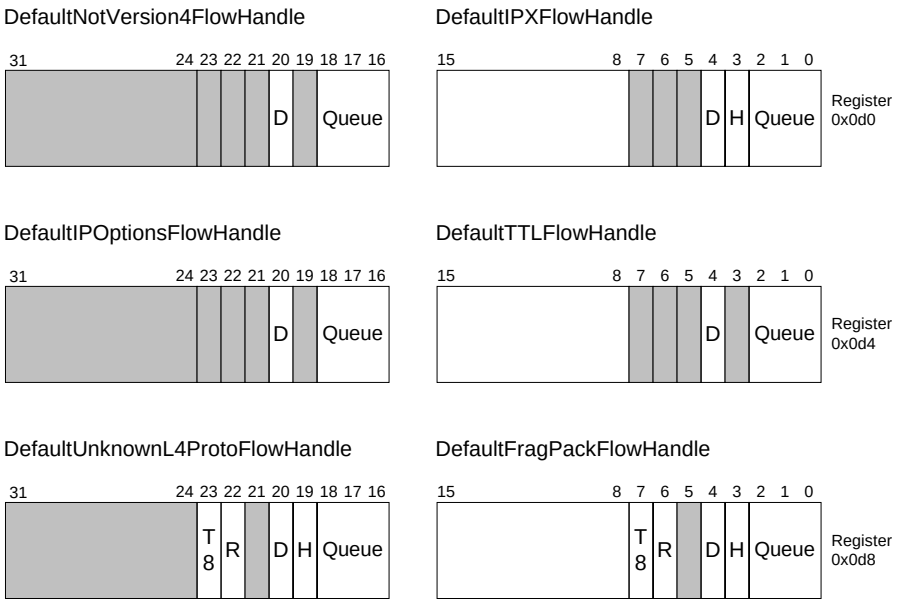


Figure A-1: Registers with Field Definitions

Appendix B: Packet Header Layouts

This section contains information for the layout of header data in IP and IPX packets.

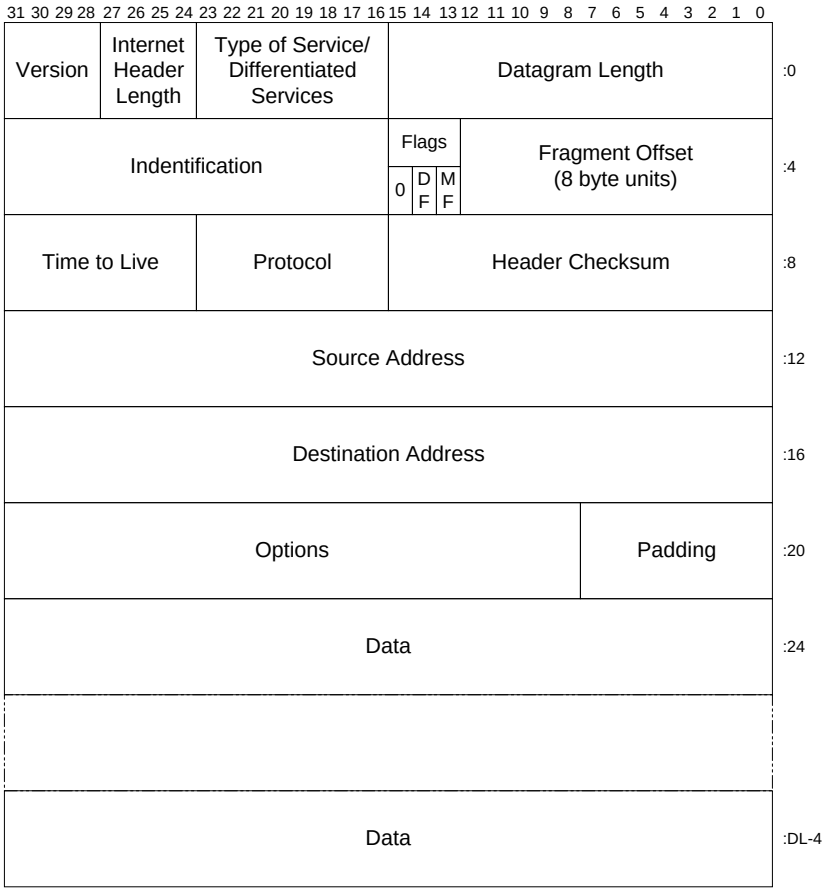


Figure B-1: IP Packet Layout

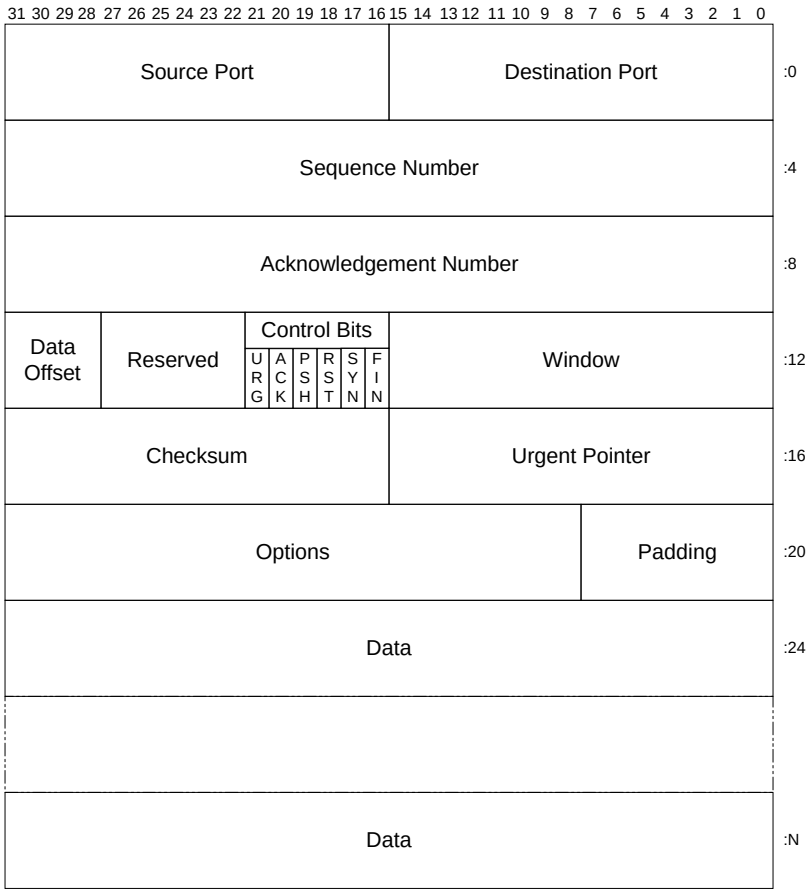


Figure B-2: TCP Packet Layout

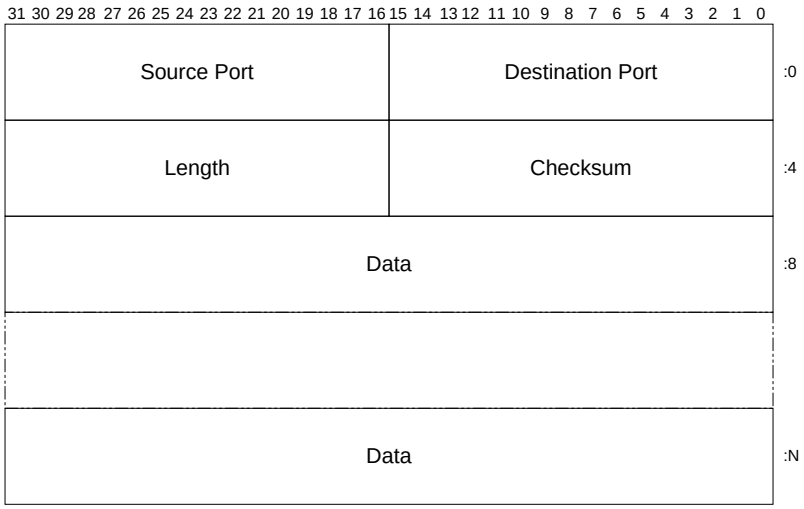


Figure B-3: UDP Packet Layout

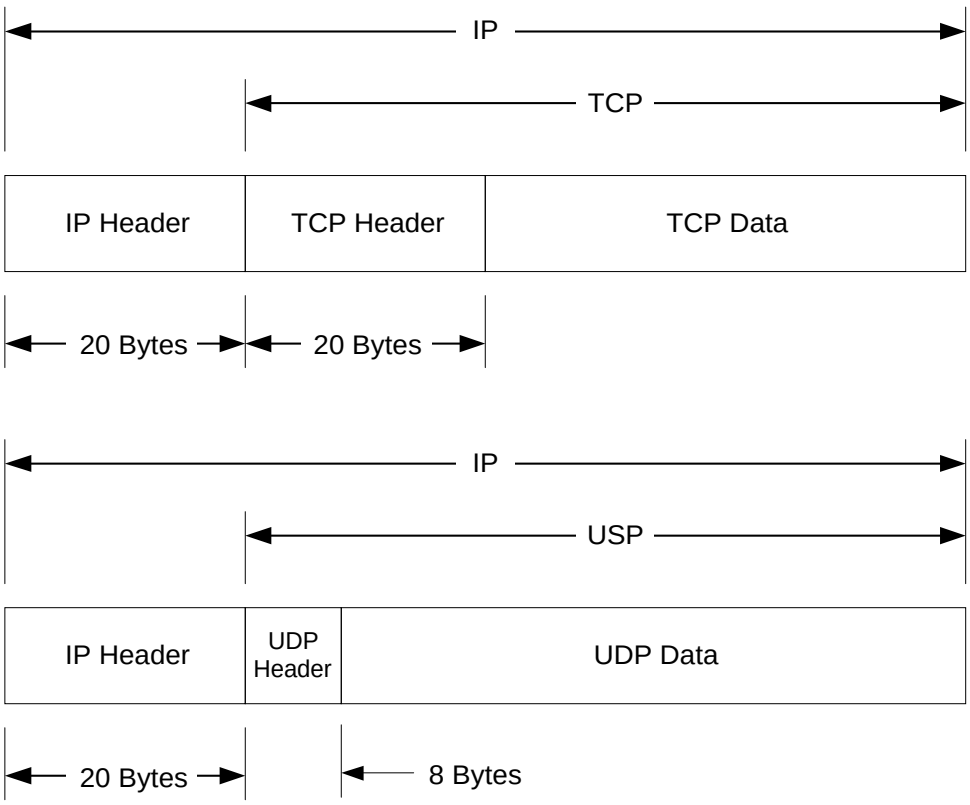


Figure B-4: TCP Data and UDP Datagram IP Encapsulation

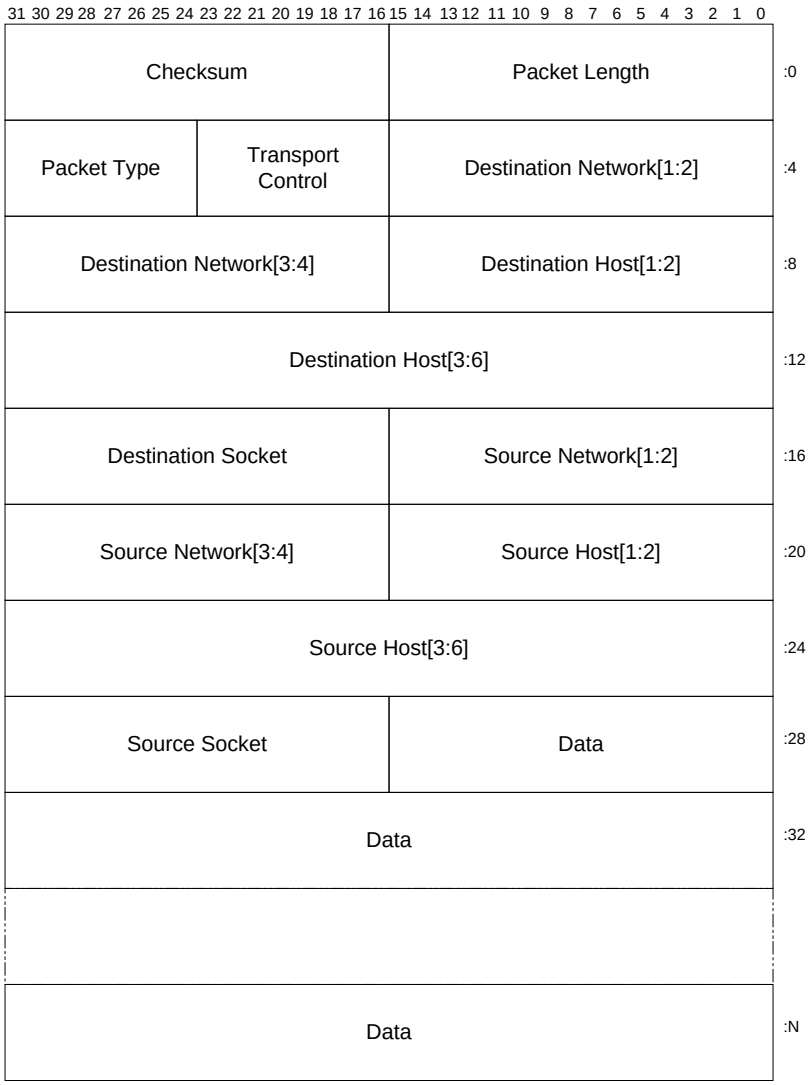


Figure B-5: IPX Packet Layout

Appendix C: C Language Header Files

Example C-1: address.h - Organized in Address Order

```
#define PSDRAMWRADDR                0x000 // R/W[19:0] Processor buffer write address
#define PSDRAMRDADDR                0x004 // R/W[19:0] Processor buffer read address
#define PSDRAMWRDATA                0x008 // R/W.16 Processor buffer write data
#define PSDRAMRDDATA                0x00c // R.16 Processor buffer read data
#define BSDRAMWRADDR                0x010 // R/W[19:0] Processor buffer pointer write address
#define BSDRAMRDADDR                0x014 // R/W[19:0] Processor buffer pointer read address
#define BSDRAMWRDATA                0x018 // R/W.32 Processor buffer pointer write data
#define BSDRAMRDDATA                0x01c // R.32 Processor buffer pointer read data
#define PORTREADY                   0x020 // R.1 Ready bit
#define PsdramMrsInitReg            0x024 // R/W[11:0] Buffer Pointer SDRAM mode register settings
#define BsdramMrsInitReg            0x028 // R/W[11:0] Buffer SDRAM mode register setting
#define BufFullThreshReg            0x02c // R/W.16 BUFFERFULL flag assertion threshold
#define BufUsedMaxThreshReg         0x030 // R/W.16 BUFFERNEARLYFULL flag
#define BufUsedMinThreshReg         0x034 // R/W.16 BUFFERNEARLYFULL flag
#define RfshMaxCntReg               0x038 // R/W.16 Refresh period
#define SYNCSUSPENDB                0x03c // R/W.1 0=Suspend SYNC pulse 1=Enable SYNC pulse Resets to 0
#define RFSHSUSPENDB                0x040 // R/W.1 0=Suspend Refresh 1=Enable Refresh Resets to 0
#define BSDRAMPCALLSTART            0x044 // R/W.1 1=Initiate Precharge All operation on Buffer SDRAM
#define PSDRAMPCALLSTART            0x048 // R/W.1 1=Initiate Precharge All operation on Buffer Pointer SDRAM
#define BsDRAMRefreshStart          0x04c // R/W.1 1=Start Refresh to Buffer Pointer Memory
#define PsDRAMRefreshStart          0x050 // R/W.1 1=Start Refresh to Pointer Memory
#define PROGSYNCREG                 0x054 // R/W[5:0] Number of clocks before the first word of TDM RX Data SYNC pulse
#define TTL_THRESH_03_02_01_00      0x060 // R/W.4*8, TTL Threshold for ports 0-3
#define TTL_THRESH_07_06_05_04      0x064 // R/W.4*8, TTL Threshold for ports 4-7
#define TTL_THRESH_11_10_09_08      0x068 // R/W.4*8, TTL Threshold for ports 8-11
#define TTL_THRESH_15_14_13_12      0x06c // R/W.4*8, TTL Threshold for ports 12-15
#define MTU_01_00                   0x070 // R/W.2*16, MTU for ports 00-01
#define MTU_03_02                   0x074 // R/W.2*16, MTU for ports 02-03
#define MTU_05_04                   0x078 // R/W.2*16, MTU for ports 04-05
#define MTU_07_06                   0x07c // R/W.2*16, MTU for ports 06-07
#define MTU_09_08                   0x080 // R/W.2*16, MTU for ports 08-09
#define MTU_11_10                   0x084 // R/W.2*16, MTU for ports 10-11
#define MTU_13_12                   0x088 // R/W.2*16, MTU for ports 12-13
#define MTU_15_14                   0x08c // R/W.2*16, MTU for ports 14-15
#define IPXattach_Port0             0x090 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port1             0x094 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port2             0x098 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port3             0x09c // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port4             0x0a0 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port5             0x0a4 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port6             0x0a8 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port7             0x0ac // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port8             0x0b0 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port9             0x0b4 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port10            0x0b8 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port11            0x0bc // R/W.32 IPX Network number per port for IPX forwarding
```

Example C-1: address.h - Organized in Address Order

```
#define IPXattach_Port12              0x0c0 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port13              0x0c4 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port14              0x0c8 // R/W.32 IPX Network number per port for IPX forwarding
#define IPXattach_Port15              0x0cc // R/W.32 IPX Network number per port for IPX forwarding
#define DefaultIPX_nv4FlowHandles     0x0d0 // R/W.2*16, (DS: DefaultIPXFlowHandle & DefaultNotVersion4FlowHandle)
#define DefaultIPopt_TTLflowHandles   0x0d4 // R/W.2*16, (DS: DefaultIPOtionsFlowHandle & DefaultTTLFlowHandle)
#define DefaultFrag_UnL4FlowHandles   0x0d8 // R/W.2*16, (DS: DefaultFragPackFlowHandle & DefaultUnknownL4ProtoFlowHandle
#define DSISEL_DSMAP_WRITE            0x0dc // W.2.16, (DS: DSisel & DSMAP)
#define SSISEL_SSMAP_READ             0x0dc // R.2.16, (DS: SSisel & SSMAP)
#define DSISEL_DSMAP_READ             0x0e0 // R.2.16, (DS: DSisel & DSMAP)
#define SSISEL_SSMAP_WRITE            0x0e0 // W.2.16, (DS: SSisel & SSMAP)
#define SIISEL_SIP                    0x0e4 // R/W.4.16, (DS: SIisel & SIP)
#define HF_NL3_DNP                    0x0e8 // R/W.4.1.16, (DS: DoNotProcess & NoL3MatchHostForward & HostForwaredPortSelect)
#define BACuFlowb_DSenable            0x0ec // R/W/2*16, (DS: BACnMicroflow & EnableDSclassifier)
#define DEF_L3AP_L4FI                 0x0f0 // R/W/2*15, (DS: L3DefaultIndexPtr & L4DefaultIndexPtr)
#define SRAMRdAddr                    0x108 // R/W[16:0] CRI SRAM read address
#define SRAMRdData                    0x10c // R.16 CRI SRAM read Data
#define SRAMWrData                    0x110 // R/W[16:0] CRI SRAM Write Data, launches the write
#define SRAMWrAddr                    0x114 // R/W.16 CRI SRAM Write Address
#define RCPRdOp                       0x118 // R/W[18:0] CRI RCP Read Operand
#define RCPRdData                     0x11c // R.32 CRI RCP Read Data
#define RCPWrData                     0x120 // R/W.32 CRI RCP Write Data, launches the write
#define RCPWrOp                       0x124 // R/W[19:0] CRI RCP Write Operand
#define RCPStatus                     0x128 // R/W[1:0] Bit 0 = RCP validity bit
#define RCPSearchData                 0x12c // R/W.32 CRI RCP Search Data, launches the search
#define RCPSearchOp                   0x130 // R/W[18:0] CRI RCP Search Operand
#define RCPSearchIndex_MF_MM          0x134 // R/W.1.1.15, (DS: RCPSearchIndex & RCPSearchMF & RCPSearchMM)
#define CREG0                         0x138 // R/W.16 1=Enable MIAN
#define QueueSRAMWriteAddress         0x140 // R/W.7 Address of Queue pointer SRAM to perform write
#define QueueSRAMWriteDataLow         0x144 // R/W.32 LOW word of write data
#define QueueSRAMWriteDataHigh        0x148 // R/W.16 HIGH word of write data
#define QueueSRAMReadAddress          0x14c // R/W.7 Address of Queue pointer SRAM to perform read
#define QueueSRAMReadDataLow          0x150 // R.32 LOW word of read data
#define QueueSRAMReadDataHigh         0x154 // R.16 HIGH word of read data
#define PacketSRAMwriteAddress        0x158 // R/W.7 Address of Packet pointer SRAM to perform write
#define PacketSRAMwriteData           0x15c // R/W.32 Packet pointer Write data
#define PacketSRAMReadAddress         0x160 // R/W.7 Address of Packet pointer SRAM to perform read
#define PacketSRAMReadData            0x164 // R.32 Packet pointer read data
#define ProcessorPortEnable           0x168 // R/W.1 1=Enable processor access to BFM and PKM memories
#define ProcessorPortReady            0x16c // R.1 Indicates the processor interface to the PKM is ready
#define NextFreePointer               0x170 // R.16 Info
#define NextPacket                    0x174 // R.16 Info
#define HeadofPacket                  0x178 // R.16 Info
#define HeadofQueue                   0x17c // R.16 Info
#define TailofQueue                   0x180 // R.16 Info
#define NumberofPacket                0x184 // R.16 Info
#define TXPacket                      0x188 // R.16 1=A packet is being TX'd
#define FullThreshold                 0x18c // R/W.16 Used packet pointer Threshold at which BUFFERFULL should be asserted
#define UpperThreshold                0x190 // R/W.16 Used packet pointer threshold at which BUFFERNEARLYFULL should be asserted
#define LowerThreshold                0x194 // R/W.16 Used packet pointer threshold at which BUFFERNEARLYFULL should be de-asserted
#define PortAlgorithmSelect           0x198 // R/W.32 Port 0-15 0 = Weighted Round Robin Mode 1 = Priority Mode
#define ResetStatus                   0x19c // R/W.32 Write 1 to bit 0 to S/W Reset Epoch
#define ReadyStatus                   0x1a0 // R.
```

Example C-1: address.h - Organized in Address Order

```
#define Version                       0x1a4 // R.32 Epoch revision number
#define CntOutOfBuffers               0x1a8 // R.16 Number of buffer full aborts
#define CntOutOfBuffersThresh         0x1ac // R/W.16 Interrupt Threshold for above
#define CntOutOfPacketPointers        0x1b0 // R.16 Number of packet pointers exhausted aborts
#define CntOutOfPacketPointersThresh  0x1b4 // R/W.16 Interrupt Threshold for above
#define CntFifoOverflow               0x1b8 // R.16 Number of multicast FIFO overflow aborts
#define CntFifoOverflowThresh         0x1bc // R/W.16 Interrupt Threshold for above
#define CntPacketTooBig               0x1c0 // R.16 Number of oversize packet aborts
#define CntPacketTooBigThresh         0x1c4 // R/W.16 Interrupt Threshold for above
#define CntPacketTooSmall             0x1c8 // R.16 Number of undersized packet aborts
#define CntPacketTooSmallThresh       0x1cc // R/W.16 Interrupt Threshold for above
#define CntL2Abort                    0x1d0 // R.16 Number of aborts from Layer 2 device
#define CntL2AbortThresh              0x1d4 // R/W.16 Interrupt Threshold for above
#define CntTTLThreshold               0x1d8 // R.16 Number of TTL threshold aborts
#define CntTTLThresholdThresh         0x1dc // R/W.16 Interrupt Threshold for above
#define CntL3FilterAbort              0x1e0 // R.16 Number of no bitmap aborts
#define CntL3FilterAbortThresh        0x1e4 // R/W.16 Interrupt Threshold for above
#define CntL4FilterAbort              0x1e8 // R.16 Number of Layer 4 filter aborts
#define CntL4FilterAbortThresh        0x1ec // R/W.16 Interrupt Threshold for above
#define CntTotalAborts                0x1f0 // R.16 Total number of packets aborted
#define CntTotalAbortsThresh          0x1f4 // R/W.16 Interrupt Threshold for above
#define Interrupt_bits                0x1f8 // R.16 Each bit is set accordingly when the interrupt condition occurs
#define Interrupt_mask                0x1fc // R/W.16 Mask for the Interrupt register
```

Table C-2: oth_code.h Header File

```
#define DIAG_INIT_PHASE                          // Diagnostics for initialization phase
#define DIAG_RUN_PHASE                           // Diagnostics when operating
#define BAC_MIAN_STRT 0x0_8000                   // Start of the BAC and MIAN tables
#define PDQ_STRT 0x1_0000                        // Start of the PDQ table
#define NUMBUFFERS 65536                         // 2^16 buffers
struct ProcAddrRng {                             // Used to define address ranges
       int AdrBeg;                               // Beginning of range
       int AdrEnd;                               // End of range
       char BlkName[3];                          // Three character block name
       int StatMask;                             // Bit in status register
};
```

Appendix D: External References

Table D-1: IETF Documents

ID	Title	Comments
RFC 791	Internet Protocol	IP
RFC 793	Transmission Control Protocol	TCP
RFC 768	User Datagram Protocol	UDP
RFC 1519	Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy	CIDR
RFC 1141, 1624	Computation of the Internet Checksum via Incremental Update	
RFC 1878	Variable Length Subnet Table for IPv4	
RFC 2474	Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers	Diff-Serv
RFC 2475	An Architecture for Differentiated Services	Diff-Serv
RFC 1042	Standard for the transmission of IP datagrams over IEEE 802 networks	IP over IEEE 802
RFC 792	Internet Control Message Protocol	ICMP
RFC 1058	Routing Information Protocol	RIP-1
RFC 2453	RIP Version 2	RIP-2
RFC 1724	RIP Version 2 MIB Extension	RIP-2
RFC 1583, 2178	OSPF Version 2	OSPF-2
RFC 1850	OSPF Version 2 Management Information Base	OSPF-2
RFC 1765	OSPF Database Overflow	OSPF
RFC 1587	The OSPF NSSA Option	OSPF
RFC 1657	Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2	BGP-4
RFC 1771	A Border Gateway Protocol 4 (BGP-4)	BGP-4
RFC 1403	BGP OSPF Interaction	
RFC 1157	Simple Network Management Protocol	SNMP
RFC 1213	Management Information Base for network management of TCP/IP-based internets: MIB-II	SNMP MIB II
RFC 1541	Dynamic Host Configuration Protocol	DHCP
RFC 1757	Remote Network Monitoring Management Information Base	RMON I
RFC 2021	Remote Network Monitoring Management Information Base Version 2 using SMIv2	RMON II

Index

http://www.music-ic.com
email: info@music-ic.com

Worldwide Headquarters
MUSIC Semiconductors
2290 N. First St., Suite 201
San Jose, CA 95131
USA
Tel: 408 232-9060
Fax: 408 232-9201
USA Only: 800 933-1550 Tech Support
888 226-6874 Product Info

Asian Headquarters
MUSIC Semiconductors
Special Export Processing Zone
Carmelray Industrial Park
Canlubang, Calamba, Laguna
Philippines
Tel: +63 49 549-1480
Fax: +63 49 549-1024
Sales Tel/Fax: +632 723-6215

European Headquarters
MUSIC Semiconductors
P. O. Box 184
6470 ED Eygelshoven
The Netherlands
Tel: +31 43 455-2675
Fax: +31 43 455-1573